GW-BASIC

Tulip®
computers
's-Hertogenbosch The Netherlands

© Copyright 1988, **Tulip Computers.**

Part number: 72-10251-01/331

Uitgiftedatum: April 1988

Niets uit deze uitgave mag worden verveelvoudigd, opgeslagen in een geautomatiseerd gegevensbestand, of openbaar gemaakt, in enige vorm of op enige wijze, hetzij elektronisch, mechanisch, door fotokopieën, opnamen, of op enige andere manier, zonder voorafgaande schriftelijke toestemming van de uitgever.

No part of this manual may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Uitgegeven door:

Tulip Computers
Postbus 3337
5203 DH 's-Hertogenbosch
Nederland

AANSPRAKELIJKHEID

De informatie in deze handleiding werd met grote zorgvuldigheid samengesteld. Er zouden echter typfouten of technische onvolkomenheden in voor kunnen komen. Tulip Computers en Microsoft Corporation aanvaarden geen verantwoordelijkheid voor schade die, direct of indirect, uit het gebruik van deze handleiding voortkomt.
De inhoud van de handleiding kan worden gewijzigd zonder voorafgaande mededeling, en impliceert geen enkele verplichting voor Tulip Computers of voor Microsoft Corporation. De in deze handleiding beschreven software is beschermd door licentie. De tekst van de licentie-overeenkomst vindt u achterin de handleiding. De software mag alleen gebruikt worden in overeenstemming met de regels van die overeenkomst. Het is wettelijk niet toegestaan deze software te kopiëren op tape, schijf, of anderszins, voor andere doeleinden dan het persoonlijk gebruik door de koper.

Handelsmerken

Tulip is een geregistreerd handelsmerk van Tulip Computers.

Microsoft en MS-DOS zijn geregistreerde handelsmerken van Microsoft Corporation.

IBM is een geregistreerd handelsmerk van International Business Machines Corporation.

DEC is een geregistreerd handelsmerk van Digital Equipment Corporation.

Uw dealer:

INHOUD

1 EERSTE KENNISMAKING MET GW-BASIC 9

Besturingssysteem 9
Inleiding 9
Notatiemethode 10
Indeling van deze handleiding 10
Informatie over BASIC 10

2 AAN DE SLAG MET GW-BASIC 11

Laden van GW-BASIC 11
Commando- en programma-modus 11
Commando-modus 12
Programma-modus 12
Opdrachtregel voor het starten van GW-BASIC 12
Veranderen van de standaardinvoer en -uitvoer 14
Opdrachten, functies, commando's en variabelen 15
Gereserveerde woorden 15
Commando's 16
Opdrachten 16
Functies 16
Variabelen 16
Structuur van programmaregels 17
Terugkeren naar MS-DOS 18

3 GW-BASIC IN DE PRAKTIJK 19

Oefeningen voor de commando-modus 19
Oefeningen voor de programma-modus 20
Functietoetsen 21
Wijzigen van regels 22
Opslaan van een programma 22

4 DE SCREEN EDITOR VAN GW-BASIC 24

Regels wijzigen in nieuwe bestanden 24
Regels wijzigen in opgeslagen bestanden 24
Wijzigen van een programmaregel 24
Toetsen met speciale functies 25
Functietoetsen 27

5 AANMAKEN EN GEBRUIKEN VAN BESTANDEN 28

Commando's voor programmabestanden 28
Gegevensbestanden 29
Aanmaken van een sequentieel bestand 29
Oproepen van een sequentieel bestand 30
Gegevens toevoegen aan een sequentieel bestand 31
Onbepaald benaderbare bestanden 31
Aanmaken van een onbepaald benaderbaar bestand 32
Oproepen van een onbepaald benaderbaar bestand 33

6 CONSTANTEN, VARIABELEN, EXPRESSIES EN OPERATOREN 36

Constanten 36
Numerieke constanten met enkele precisie en met dubbele precisie 37
Variabelen 37
Namen en type-aanduidingen van variabelen 38
Tekens voor type-aanduidingen 38
MATRIX variabelen 39
Vereiste geheugenruimte voor het opslaan van 39
Type conversie 40
Expressies en operatoren 41
Wiskundige operatoren 41
Relationele operatoren 43
Logische operatoren 43
Functionele operatoren 45
Tekenreeks-operatoren 45

7 COMMANDO'S, OPDRACHTEN EN FUNCTIES 47

ABS functie 48
ASC functie 48
ATN functie 49
AUTO commando 49

BEEP opdracht 50
BLOAD opdracht 50
BSAVE commando 51

CALL opdracht 52
CDBL functie 54
CHAIN opdracht 55
CHDIR commando 56
CHR$ functie 56
CINT functie 57
CIRCLE opdracht 57
CLEAR commando 59
CLOSE opdracht 60
CLS opdracht 60
COLOR opdracht 61
COM(n) opdracht 63
COMMON opdracht 64
CONT commando 65
COS functie 65
CSNG functie 66
CSRLIN variabele 66
CVI, CVS, CVD functies 67

DATA opdracht 67
DATE$ opdracht en variabele 69
DEF FN opdracht 69
DEFINT, DEFSNG, DEFDBL, DEFSTR opdrachten 71
DEF SEG opdracht 72
DEF USR opdracht 73
DELETE commando 73
DIM opdracht 74
DRAW opdracht 75

EDIT commando 78
END opdracht 78
ENVIRON opdracht 79
ENVIRON$ functie 80
EOF functie 82

ERASE opdracht 82
ERDEV, ERDEV$ variabelen 83
ERROR opdracht 83
EXP functie 85
EXTERR functie 85

FIELD opdracht 86
FILES commando 87
FIX functie 88
FOR...NEXT opdrachten 88
FRE functie 90

GET opdracht (Bestanden I/O) 91
GET opdracht (Grafisch) 91
GOSUB...RETURN opdrachten 93
GOTO opdracht 95

HEX$ functie 95

IF opdracht 96
INKEY$ variabele 97
INP functie 98
INPUT opdracht 99
INPUT# opdracht 100
INPUT$ functie 101
INSTR functie 102
INT functie 103
IOCTL opdracht 104
IOCTL$ functie 104

KEY opdracht 105
KEY(n) opdracht 107
KILL commando 107

LEFT$ functie 108
LEN functie 108
LET opdracht 109
LINE opdracht 110
LINE INPUT opdracht 112
LINE INPUT# opdracht 113
LIST commando 113
LLIST commando 114
LOAD commando 115
LOC functie 115
LOCATE opdracht 116
LOCK opdrachten 117
LOF functie 118
LOG functie 119
LPOS functie 119
LPRINT en LPRINT USING opdrachten 120
LSET en RSET opdrachten 120

MERGE commando 121
MID$ functie 121
MID$ opdracht 122
MKDIR commando 123
MKI$, MKS$, MKD$ functies 123

NAME commando 124
NEW commando 124

OCT$ functie 125
ON COM(n), ON KEY(n), ON PEN, ON PLAY(n), ON STRIG(n),
en ON TIMER(n) opdrachten 125
Aanvullende informatie voor elk type detectie 127
ON COM(n) 129
ON ERROR GOTO opdracht 130
ON...GOTO en ON...GOSUB opdrachten 131
ON KEY(n) opdracht 132
ON PEN opdracht 134
ON PLAY(n) opdracht 135
ON STRIG(n) opdracht 136
ON TIMER(n) opdracht 137
OPEN opdracht 138
OPEN COM(n) opdracht 142
OPTION BASE opdracht 144
OUT opdracht 144

PAINT opdracht 145
PALETTE, PALETTE USING opdrachten 147
PCOPY commando 149
PEEK functie 150
PEN opdracht en functie 150
PLAY opdracht 152
PLAY(n) functie 153
PMAP functie (Grafisch) 154
POINT functie 154
POKE opdracht 155
POS functie 156
PRESET en PSET opdrachten 156
PRINT opdracht 158
PRINT USING opdracht 159
PRINT# en PRINT# USING opdrachten 162
PUT opdracht (Grafisch) 164
PUT opdracht (Bestanden) 165

RANDOMIZE opdracht 166
READ opdracht 168
REM opdracht 169
RENUM commando 170
RESET commando 171
RESTORE commando 171
RESUME opdracht 172
RETURN opdracht 172
RIGHT$ functie 173
RMDIR functie 173
RND functie 174
RUN commando 174

SAVE commando 175
SCREEN functie 176
SCREEN opdracht 177
SGN functie 181
SHELL opdracht 182
SIN functie 183
SOUND opdracht 183
SPACE$ functie 185
SPC functie 185
SQR functie 186
STICK functie 186
STOP opdracht 187
STR$ functie 188
STRIG opdracht en functie 188

STRIG(n) opdracht 189
STRING$ functie 190
SWAP opdracht 190
SYSTEM commando 191

TAB functie 191
TAN functie 192
TIME$ opdracht en variabele 192
TIMER functie 193
TRON en TROFF commando's 194

UNLOCK opdracht 194
USR functie 196

VAL functie 196
VARPTR functie 197
VARPTR$ functie 199
VIEW opdracht 200
VIEW PRINT opdracht 201

WAIT opdracht 201
WHILE...WEND opdracht 202
WIDTH opdracht 203
WINDOW opdracht 205
WRITE opdracht 206
WRITE# opdracht 207

Appendix A - Foutcodes en foutmeldingen 209

Appendix B - Wiskundige functies 215

Appendix C - ASCII-codes 216

Appendix D - Subroutines in assembleertaal 219

Reserveren van geheugenruimte 219
CALL-opdracht 220
USR-functie 223
Programma's voor het oproepen van assembleerroutines 225

Appendix E - BASIC-programma's omzetten in GW-BASIC 227

Tekenreeksdimensies 227
Meervoudige instructies 228
Meervoudige opdrachten 228
MAT-functies 228
FOR...NEXT 228

Appendix F - Communicatie 229

De INPUT$-functie 230
GET en PUT opdrachten voor COM-bestanden 231

Appendix G - Equivalenten van hexadecimale waarden 235

Appendix H - Scan-codes voor toetsen 237

Appendix I - Door GW-BASIC herkende tekens 239

Woordenlijst 240

Register 257

1 EERSTE KENNISMAKING MET GW-BASIC

Microsoft® GW-BASIC® is een eenvoudige en gebruiksvriendelijke programmeertaal die snel aan te leren is. De in GW-BASIC gebruikte opdrachten en wiskundige notaties vertonen veel verwantschap met normaal Engels. Met GW-BASIC kunt u uw computer zowel simpele als complexe programma's laten uitvoeren. Bovendien is het mogelijk bestaande GW-BASIC programma's aan te passen aan uw eigen wensen.

Deze handleiding is bedoeld voor het gebruik van de GW-BASIC Interpreter in combinatie met het MS-DOS® besturingssysteem (operating system).

Besturingssysteem

Voor deze versie van GW-BASIC moet u beschikken over MS-DOS 3.2 of een latere versie.

Inleiding

De GW-BASIC-bestanden worden opgeslagen op uw MS-DOS diskettes. Zorg dat u een kopie maakt van deze diskette voordat u aan het werk gaat met GW-BASIC. Gebruik deze kopie als werk-diskette.

Opmerking:

Deze handleiding is bestemd voor de gebruiker die bekend is met het MS-DOS besturingssysteem. Raadpleeg uw MS-DOS handleiding voor het gebruik van dit besturingssysteem.

Notatiemethode

In deze handleiding worden de volgende notaties gebruikt om bepaalde elementen in de tekst te benadrukken:

vet gedrukte tekst
Wordt gebruikt voor commando's, opties, optieletters en tekst die letterlijk moet worden ingetikt zoals deze is aangegeven.

cursief gedrukte tekst
Wordt gebruikt voor bestandsnamen, variabelen en argumenten die aangeven wat de gebruiker moet intikken.

`typemachinetekst`
Dit teksttype wordt gebruikt voor voorbeelden.

KLEINE HOOFDLETTERS
Op deze wijze worden toetsen of combinaties van toetsen aangeduid.

[ ]
Worden gebruikt om in de beschrijving van commando's argumenten aan te geven die optioneel zijn.

Indeling van deze handleiding

De GW-BASIC-handleiding is ingedeeld in zeven hoofdstukken, negen appendices en een verklarende woordenlijst. Hieronder wordt de inhoud van de zeven hoofdstukken in het kort weergegeven.

Hoofdstuk 1, "Eerste kennismaking met GW-BASIC", geeft een aantal algemene inleidende opmerkingen.

Hoofdstuk 2, "Aan de slag met GW-BASIC", geeft de basisprincipes van het programmeren.

Hoofdstuk 3, "GW-BASIC in de praktijk", legt uit hoe u de principes uit hoofdstuk 2 praktisch kunt toepassen.

Hoofdstuk 4, "De screen editor van GW-BASIC", bespreekt de commando's die worden gebruikt bij het invoeren of aanpassen van een GW-BASIC programma. Dit hoofdstuk geeft tevens een overzicht van de unieke eigenschappen van de tien programmeerbare functietoetsen, toetsencombinaties en alle andere speciale toetsen.

Hoofdstuk 5, "Aanmaken en gebruiken van bestanden", legt uit hoe bestanden moeten worden aangemaakt en hoe u de invoer/uitvoer (I/O) operaties van de diskette moet gebruiken.

Hoofdstuk 6, "Constanten, variabelen, expressies en operatoren", beschrijft het gebruik van constanten, variabelen, expressies en operatoren in GW-BASIC.

Hoofdstuk 7, "Commando's, opdrachten en functies" geeft een beschrijving van de commando's, opdrachten en functies, inclusief duidelijke voorbeelden.

Informatie over BASIC

Deze handleiding geeft een complete uitleg over het gebruik van Microsoft BASIC, maar bevat geen oefeningen om in BASIC te leren programmeren. Als u nooit eerder met BASIC hebt gewerkt, dan raden wij u aan eerst een boek over het programmeren in BASIC door te nemen. Er zijn op dit gebied veel publicaties verschenen.

2 AAN DE SLAG MET GW-BASIC

In dit hoofdstuk wordt uitgelegd hoe u GW-BASIC in het geheugen kunt laden. Ook wordt een beschrijving gegeven van twee soorten bedienings-modi, de opdrachtregel waarmee u GW-BASIC start, en van de verschillende onderdelen van GW-BASIC.

Laden van GW-BASIC

Voordat u aan de slag kunt met GW-BASIC moet u dit programma, vanaf de MS-DOS-werk-diskette, in het geheugen van uw computer opslaan. Ga hierbij als volgt te werk:

1 Schakel uw computer in.

2 Plaats de MS-DOS-werk-diskette in station A van uw computer en druk op ENTER.

3 Op het scherm verschijnt de prompt (systeemvraag) A>. Tik vervolgens het onderstaande commando in en druk daarna op ENTER:

```
gwbasic
```

Na het intikken van GW-BASIC, wordt de MS-DOS prompt A> vervangen door de GW-BASIC-prompt Ok.

De regel `"XXXXX Bytes vrij"` geeft aan hoeveel bytes geheugenruimte nog vrij zijn voor GW-BASIC.

De functietoetsen F1 t/m F10 verschijnen onderin het beeldscherm. Deze functietoetsen vervangen een aantal toetshandelingen en kunnen u veel tijd besparen. Hoofdstuk 4, "De screen editor van GW-BASIC" geeft meer informatie over de functietoetsen.

Commando- en programma-modus

Na het initialiseren (laden) van GW-BASIC verschijnt de prompt Ok. Dit duidt aan dat GW-BASIC in de *commando-modus* is. Dit houdt in dat u commando's kunt intikken die direct worden uitgevoerd. U kunt nu doorgaan in de *commando-modus*, of overschakelen naar de *programma-modus*.

COMMANDO-MODUS

In de commando-modus worden GW-BASIC-opdrachten en -commando's uitgevoerd zoals u ze hebt ingetikt. Resultaten van wiskundige en logische operaties kunnen direct worden weergegeven en/of worden opgeslagen voor later gebruik. Na het weergeven of opslaan van de resultaten raken de commando's zelf verloren. Deze modus is erg nuttig bij het verbeteren van fouten en bij het uitvoeren van snelle berekeningen waarvoor geen compleet programma nodig is.

PROGRAMMA-MODUS

De programma-modus wordt gebruikt bij het invoeren van complete programma's. Elke regel van het programma begint met een nummer en wordt in het geheugen opgeslagen. Als u het commando RUN geeft, wordt het programma uitgevoerd.

Opdrachtregel voor het starten van GW-BASIC

Met de opdrachtregel van GW-BASIC kunt u de modus en de voorwaarden tijdens het gebruik van GW-BASIC wijzigen.

Opmerking:

Als u de modus GW-BASIC wilt wijzigen, verander dan niets aan de volgorde van de argumenten in de opdrachtregel, zoals die in onderstaande syntaxis is aangegeven. Plaats een komma als u een argument wilt overslaan. Hierdoor weet de computer dat u dit argument niet hebt gewijzigd.

De opdrachtregel van GW-BASIC heeft de volgende syntaxis:

gwbasic[*bestandsnaam*][*<stdin*][[>]*>stdout*][/f:*n*][/i][/s:*n*][/c:*n*][/m:[*n*][,*n*]][/d]

Het argument *bestandsnaam* is de naam van een programmabestand van GW-BASIC. Als dit argument wordt gebruikt, gaat GW-BASIC te werk alsof een RUN-commando is gegeven. Als de bestandsnaam geen bestandstype heeft, wordt het standaard bestandstype .BAS aangenomen. Dit bestandstype wordt gebruikt voor bestanden van GW-BASIC. Een bestandsnaam mag maximaal acht tekens lang zijn, aangevuld met een decimale punt en drie tekens voor het bestandstype.

Het argument *<stdin* geeft aan dat de standaardinvoer niet van het toetsenbord afkomstig is, maar van het opgegeven bestand. Het argument *<stdin* moet altijd vóór de optieletters worden geplaatst. Als het programma meerdere bestanden zou kunnen gebruiken, kunt u met behulp van dit commando aangeven welk bestand moet worden gebruikt.

Het argument *>stdout* geeft aan dat de uitvoer van GW-BASIC naar het opgegeven randapparaat of het opgegeven bestand moet worden gestuurd. Dit argument moet altijd vóór de optieletters worden geplaatst. Als u vóór *stdout* *>>* intikt, wordt de uitvoer aan het einde van het bestand toegevoegd.

U kunt GW-BASIC voor het lezen weer verwijzen naar de standaardinvoer (toetsenbord) of voor het schrijven naar de standaarduitvoer (beeldscherm). Geef hiertoe de bestandsnamen van de invoer en uitvoer als volgt aan in de opdrachtregel:

gwbasic *programmanaam <invoer bestand[>]>uitvoer bestand*

Na de bespreking van de opdrachtregel van GW-BASIC, wordt het verwijzen naar een ander bestand uitgelegd.

In de opdrachtregel komen diverse opties voor. Deze geven aan hoe het programma GW-BASIC moet worden uitgevoerd. Door het wijzigen van deze opties kiest u een andere dan de standaardinstelling. Een dergelijke optie wordt voorafgegaan door een (/).

De optie /f:*n* bepaalt het maximale aantal bestanden dat tegelijkertijd geopend mag zijn tijdens het gebruik van GW-BASIC. Elk bestand vereist 194 bytes voor het File Control Block (FCB) en 128 bytes voor de buffer. U kunt de grootte van de buffer wijzigen met de optie /s:. Als u de optie /f: weglaat, blijft het standaard aantal geopende bestanden (3) gelden. Als de optie /i niet in de opdrachtregel is aangegeven, wordt de optie /f: genegeerd.

De optie /i reserveert geheugenruimte voor bestand-operaties die zijn gebaseerd op de opties /s en /f.

De optie /s:*n* bepaalt de maximale record-lengte. De optie *rec-lengte* in de opdracht OPEN mag deze waarde niet overschrijden. Als de optie /s: niet in de opdrachtregel is aangegeven, blijft de standaard record-lengte van 128 bytes gelden. De maximale record-lengte is 32767 bytes.

De optie /c:*n* bestuurt communicatie via de beide RS-232 interfaces. Als de RS-232-kaarten zijn ingebouwd en in de opdrachtregel /c:0 wordt aangegeven, dan schakelt u de communicatie- en I/O-mogelijkheden van deze kaarten uit. Als de optie /c: niet in de opdrachtregel is aangegeven, worden per kaart 256 bytes gereserveerd voor de ontvangstbuffer en 128 bytes voor de verzendbuffer. Als uw systeem geen RS-232-kaarten bezit, heeft de optie /c: uiteraard geen enkel effect.

De optie /m:*n*[,*n*] bepaalt het grootste geheugenadres (eerste *n*) en de maximale blokgrootte (tweede *n*) voor GW-BASIC. GW-BASIC reserveert 64 Kbyte geheugen voor het data-segment (DS) en het stack-segment. Als u bij GW-BASIC machinetaal-routines wilt gebruiken, moet u via de optie /m: het grootst mogelijke adres instellen. De maximale grootte van een blok wordt weergegeven in veelvouden van 16. Deze wordt gebruikt om naast de werkruimte voor GW-BASIC plaats te reserveren voor de assembleerroutines.

Het grootste geheugenadres fungeert tevens als standaard maximale blokgrootte. De standaardwaarde voor het grootste geheugenadres is 64 Kbyte. Als de maximale blokgrootte niet is opgegeven, kiest de computer de standaard maximale blokgrootte (in veelvouden van 16).

Met de optie /d activeert u de dubbele precisie. Als deze optie wordt geactiveerd, worden ongeveer 3000 bytes extra geheugenruimte in beslag genomen. Deze optie heeft effect op de volgende functies: ATN, COS, EXP, LOG, SIN, SQR en TAN.

Opmerking:

Alle getallen voor het instellen van opties in de opdrachtregel mogen zowel in decimale, octale (voorafgegaan door &O) als hexadecimale (voorafgegaan door &H) vorm worden opgegeven.

Hieronder volgt een aantal voorbeelden van GW-BASIC-opdrachtregels:

Deze regel gebruikt drie bestanden en neemt 64 Kbytes in beslag. Het programmabestand *boekhoud.bas* wordt geladen en uitgevoerd:

```
A>gwbasic BOEKHOUD
```

Deze regel gebruikt zes bestanden en neemt 64 Kbytes in beslag. Het programmabestand *invent.bas* wordt geladen en uitgevoerd:

```
A>gwbasic INVENT /F:6
```

De volgende instructie maakt RS-232-communicatie onmogelijk. De regel gebruikt alleen de eerste 32 Kbyte van het geheugen. De resterende 32 Kbyte wordt gereserveerd voor programma's:

```
A>gwbasic /C:0 /M:32768,4096
```

De volgende regel gebruikt vier bestanden en staat een maximale record-lengte toe van 512 bytes:

```
A>gwbasic /F:4 /S:512
```

De volgende regel gebruikt 64 Kbyte geheugen en drie bestanden. Voor de RS-232 ontvangstbuffers worden 512 bytes gereserveerd, voor de verzendbuffers 128 bytes. Het programmabestand TTY.BAS wordt geladen en uitgevoerd:

```
A>gwbasic TTY /C:512
```

Appendix F geeft meer informatie over RS-232-communicatie.

VERANDEREN VAN DE STANDAARDINVOER EN -UITVOER

Als is opgegeven dat de invoer niet gelezen wordt vanaf het toetsenbord, worden alle INPUT, LINE INPUT, INPUT$ en INKEY$ opdrachten gelezen vanaf het opgegeven invoerbestand.

Als ook de uitvoer wordt aangepast, wordt de inhoud van PRINT opdrachten naar het opgegeven uitvoerbestand gestuurd in plaats van naar het beeldscherm.

Foutmeldingen worden behalve in het opgegeven bestand ook nog steeds via het beeldscherm weergegeven.

Het bestand invoer van KYBD: wordt nog steeds gelezen vanaf het toetsenbord.

Het bestand uitvoer naar SCRN: wordt nog steeds naar het beeldscherm gestuurd.

Als de ON KEY(*n*) opdracht wordt gebruikt, blijft GW-BASIC geprogrammeerde toetsen detecteren.

Als u hebt opgegeven dat de uitvoer niet op de standaardmanier moet worden weergegeven en vervolgens op CTRL-BREAK drukt, worden alle open bestanden gesloten. Op het beeldscherm verschijnt vervolgens de melding "Onderbroken in nnnn" en u gaat vanuit GW-BASIC naar MS-DOS.

Als u een andere dan de standaardinvoer hebt opgegeven, blijft GW-BASIC deze hanteren tot CTRL-Z wordt gedetecteerd. Deze conditie kan worden getest met de functie End-Of-File (EOF). Als het bestand niet met CTRL-Z is afgesloten, of als een GW-BASIC-instructie probeert informatie in te lezen terwijl het einde van het bestand al werd bereikt, dan worden alle open bestanden gesloten, en keert het systeem terug naar MS-DOS.

Meer informatie over deze en andere opdrachten, functies, commando's en in deze tekst genoemde variabelen kunt u vinden in hoofdstuk 7.

Enkele voorbeelden van gewijzigde invoer en uitvoer:

```
GW-BASIC PROG >DATA.OUT
```

INPUT en LINE INPUT opdrachten blijven gegevens lezen vanaf het toetsenbord. PRINT opdrachten sturen gegevens naar het bestand *data.out*.

```
gwbasic PROG1 <DATA.IN
```

De INPUT en LINE INPUT opdrachten lezen gegevens uit het bestand *data.in*. De PRINT opdrachten sturen gegevens naar het beeldscherm.

```
gw.basic MYPROG <GRAFISCH.DAT >AFBEELD.DAT
```

De INPUT en LINE INPUT opdrachten lezen gegevens uit het bestand *grafisch.dat*. PRINT opdrachten sturen gegevens naar het bestand *afbeeld.dat*.

```
gwbasic MYPROG <\STAT\JAN\TRANS.DAT >>\STAT\STAT.DAT
```

De INPUT en LINE INPUT opdrachten lezen gegevens vanaf het bestand *\stat\jan\trans.dat*. De PRINT opdracht stuurt gegevens naar het bestand *\stat\stat.dat*.

Opdrachten, functies, commando's en variabelen

Een GW-BASIC programma bestaat uit verschillende elementen: gereserveerde woorden, commando's, opdrachten, functies en variabelen.

GERESERVEERDE WOORDEN

De gereserveerde woorden van GW-BASIC, zoals PRINT, GOTO en RETURN, hebben voor de GW-BASIC-interpreter een speciale betekenis. GW-BASIC interpreteert deze woorden als een deel van opdrachten of commando's.

Als u gereserveerde woorden gebruikt als namen van variabelen, interpreteert het systeem deze variabelen als commando's. Gereserveerde woorden mogen echter wel deel uitmaken van variabelenamen.

Deze woorden worden in het systeem opgeslagen als *tokens* (tekens van 1- of 2-bytes), zodat ze relatief weinig geheugenruimte in beslag nemen.

COMMANDO'S

Commando's en opdrachten zijn uitvoerbare instructies. Commando's worden meestal uitgevoerd in de commando-modus van de interpreter. In het algemeen worden commando's gebruikt voor onderhoud zoals het wijzigen, het laden of het opslaan van programma's. Als na het oproepen van GW-BASIC de prompt *Ok* verschijnt, is de commando-modus geactiveerd.

OPDRACHTEN

Een opdracht, zoals ON ERROR...GOTO, bestaat uit een aantal gereserveerde woorden in een programmaregel. Bij het uitvoeren van het programma worden de opdrachten uitgevoerd zoals ze zijn ingevoerd.

FUNCTIES

De interpreter van GW-BASIC voert numerieke en tekenreeks-functies uit.

Numerieke functies

De GW-BASIC-interpreter voert bepaalde wiskundige (rekenkundige of algebraïsche) berekeningen uit. Bijvoorbeeld het berekenen van de sinus (sin), cosinus (cos) en de tangens (tan) van een hoek.

Wanneer geen dubbele precisie is opgegeven, worden de resultaten van numerieke functies weergegeven als gehele getallen (integers) of als getallen met enkele precisie.

Tekenreeks-functies

Tekenreeks-functies voeren bewerkingen op tekenreeksen uit. Zo geven de functies TIME$ en DATE$ de tijd en de datum aan. Als bij het opstarten van het systeem de juiste tijd en datum zijn ingevoerd, geeft deze functie beide grootheden weer (de interne klok van de computer houdt de tijd bij).

Door de gebruiker zelf gedefinieerde functies

Met behulp van de opdracht DEF FN kan de gebruiker zelf functies (tekenreeks-functies of numerieke functies) definiëren.

VARIABELEN

In variabelen kunt u combinaties van alfanumerieke tekens opslaan. De waarde van variabelen kan door uitgevoerde bewerkingen steeds veranderen. Door de variabelenaam in instructies te gebruiken, kunt u de waarde van die variabele laten lezen, veranderen of weergeven.

GW-BASIC kent enkele variabelen met een voorgeprogrammeerde taak. Zo geeft ERR de laatst voorgekomen fout in het programma weer. ERL geeft de positie van de fout aan. Variabelen kunnen worden gedefinieerd of gewijzigd door de gebruiker of door de inhoud van het programma.

De commando's, opdrachten, functies en variabelen worden beschreven in hoofdstuk 7.

Structuur van programmaregels

Onderdelen van een programma worden opdrachten genoemd, die worden samengesteld uit afzonderlijke elementen. Opdrachten vertonen veel verwantschap met normale Engelse zinnen. Opdrachten die op een logische wijze zijn samengevoegd vormen een programma. In hoofdstuk 7 wordt een overzicht gegeven van alle beschikbare opdrachten.

De programmaregels van GW-BASIC hebben het volgende opzet:

nnnnn opdracht[opdrachten]

waarbij *nnnnn* een regelnummer is,

waarbij *opdracht* een GW-BASIC-opdracht is.

Elke programmaregel moet met een regelnummer beginnen. Dit nummer moet een geheel getal zijn tussen 0 en 65529. Gewoonlijk wordt gebruik gemaakt van getallen als 10, 20, 30 en 40. Hierdoor is het mogelijk achteraf nog extra regels toe te voegen. Aangezien de computer opdrachten uitvoert in de volgorde van de regelnummers hoeft u de later toegevoegde regels niet op de juiste plaats in te tikken. Als u bijvoorbeeld regel 35 intikt achter regel 60, voegt GW-BASIC die regel op juiste plaats in en voert hem uit na regel 30 en voor regel 40. Hierdoor is het niet nodig een geheel nieuw programma in te voeren als u een regel wilt tussenvoegen.

Een programmaregel moet na het regelnummer minimaal één teken bevatten en mag hoogstens 255 tekens lang zijn. Met de regelnummers geeft u de volgorde aan waarin de regels in het geheugen worden opgeslagen. Deze nummers worden ook gebruikt als verwijzingen in opdrachten waarmee een sprong naar een andere regel wordt gemaakt (zie bijvoorbeeld GOTO) en bij het veranderen van een programma. U sluit een opdrachtregel af door op ENTER te drukken.

Een regel mag meerdere opdrachten bevatten. De afzonderlijke opdrachten worden van elkaar gescheiden door een dubbele punt (:). De programmaregel mag eventueel meerdere schermregels lang zijn, maar nooit de maximale lengte van 255 tekens overschrijden.

De maximale regellengte van uw scherm is 80 tekens. Als een opdracht langer is dan 80 tekens springt de cursor automatisch naar de volgende regel. Geef het einde van de regel aan door op ENTER te drukken. Gebruik de ENTER-toets niet als de cursor het einde van de schermregel nadert. Als u op CTRL-ENTER drukt, springt de cursor naar de volgende regel van het beeldscherm zonder dat hiermee het begin van een nieuwe opdrachtregel wordt aangeduid. Als u alleen op ENTER drukt, wordt de afgeronde opdrachtregel in het programma opgeslagen.

In GW-BASIC wordt elke tekstregel die met een cijfer begint, beschouwd als een programmaregel. Een regel kan op drie manieren worden opgesteld:

- Toevoegen van een nieuwe regel aan het programma. Een regel is alleen geldig als deze begint met een toegestaan regelnummer (een geheel getal tussen 0 en 65529) en verder minstens één alfanumeriek of speciaal teken bevat.
- Wijzigen van een bestaande regel. U kunt alleen een regel wijzigen die een bestaand regelnummer heeft. De gewijzigde regel vervangt de regel in het programma. Dit wordt ook wel "*edit*" genoemd.

Opmerking:
Als u een nieuwe regel toevoegt met een nummer dat al voor een andere programmaregel werd gebruikt, wordt de informatie van de originele regel gewist. Wees daarom voorzichtig met het invoeren van programmaregels in de commando-modus.

- Wissen van een bestaande regel. Als u een nummer van een bestaande regel intikt en vervolgens op ENTER drukt, wordt de inhoud van die regel gewist. Als u probeert een regel te wissen en daarbij een niet bestaand regelnummer opgeeft, verschijnt de foutmelding "Ongedefinieerd regelnummer".

Terugkeren naar MS-DOS

Sla eerst alle gegevens op die u in GW-BASIC hebt ingevoerd, voordat u terugkeert naar MS-DOS. Als u dit niet doet gaan deze gegevens verloren. Ga daarna als volgt te werk:

Tik achter de prompt Ok het volgende commando in:

```
system
```

Druk daarna op ENTER. Het systeem keert nu terug naar MS-DOS en de prompt A> verschijnt op uw beeldscherm.

3 GW-BASIC IN DE PRAKTIJK

Dit hoofdstuk bevat oefeningen aan de hand waarvan de tot nu toe behandelde theorie in de praktijk kan worden gebruikt. Dit is het juiste moment om uw computer in te schakelen en de GW-BASIC-interpreter te laden (als u dit nog niet hebt gedaan).

Oefeningen voor de commando-modus

In de commando-modus kunt u de rekenkundige basisbewerkingen uitvoeren. GW-BASIC herkent de volgende symbolen als wiskundige operatoren:

Bewerking	Operator
Optellen	+
Aftrekken	-
Vermenigvuldigen	*
Delen	/

U voert een bewerking in door achter Ok een vraagteken (?) in te tikken. U kunt in GW-BASIC het gereserveerde woord PRINT altijd vervangen door een vraagteken. Tik vervolgens de gewenste bewerking in en druk ENTER. Het antwoord verschijnt dan op het beeldscherm.

Tik het volgende opdracht in en druk op ENTER:

```
?2+2
```

Het antwoord verschijnt dan op uw beeldscherm:

```
?2+2
4
Ok
```

Als u met andere rekenkundige bewerkingen wilt oefenen, vervang dan het plusteken door een andere operator.

Met GW-BASIC kunt u ook ingewikkelde algebraïsche en trigonometrische bewerkingen uitvoeren. In hoofdstuk 6 vindt u de juiste methode voor het toepassen van deze functies.

Oefeningen voor de programma-modus

Met GW-BASIC kunnen naast eenvoudige algebraïsche bewerkingen ook andere functies worden uitgevoerd. U kunt ook programma's opstellen waarbij eerst een aantal bewerkingen wordt uitgevoerd en vervolgens het antwoord wordt weergegeven. Stel voordat u begint te programmeren een aantal opdrachten op. Vergeet niet dat een regel meerdere opdrachten kan bevatten, en dat iedere regel moet beginnen met een nummer.

Als u het commando PRINT 2+3 als een opdracht wilt invoeren, tik dan het volgende in:

```
10 print 2+3
```

Als u nu op ENTER drukt, springt de cursor naar de volgende regel. Verder gebeurt er niets. Als u deze berekening wilt laten uitvoeren, tik dan het volgende commando in:

```
run
```

Als u vervolgens op ENTER drukt, verschijnt de volgende informatie op uw beeldscherm:

```
Ok
10 print 2+3
run
5
Ok
```

U hebt zojuist uw eerste programma in GW-BASIC geschreven.

De computer voert het programma pas uit als u het commando RUN geeft. Hierdoor is het mogelijk meerdere instructieregels op te geven.

Het volgende programma bestaat uit twee instructieregels. Tik deze regels in:

```
10 X=3
20 print 2+X
```

Geef het commando RUN om de berekening te laten uitvoeren.

Dan verschijnt dit op uw beeldscherm:

```
Ok
10 X=3
20 print 2+X
run
5
Ok
```

Een programma voor het uitvoeren van een berekening onderscheidt zich op twee manieren van een berekening tijdens de commando-modus:

1 De regels zijn genummerd.

2 Het gebruik van het RUN-commando.

Deze kenmerken maken de computer duidelijk dat alle opdrachten zijn ingevoerd en dat het programma van het begin tot het einde kan worden uitgevoerd. Aan de regelnummering kan de computer zien dat het een programma is in plaats van een instructie in de commando-modus en dat de uitvoering van de instructies pas na het RUN-commando mag plaatsvinden.

In de commando-modus worden instructies direct uitgevoerd; in de programma-modus worden programma's uitgevoerd na een RUN commando.

Als u het LIST commando geeft en vervolgens op ENTER drukt, wordt het gehele programma weergegeven:

```
list
```

Op het beeldscherm verschijnt dan:

```
Ok
10 X=3
20 print 2+X
run
Ok
5
Ok
list
10 X=3
20 PRINT 2+X
Ok
```

Zoals u ziet, is er in het programma iets gewijzigd. De door u ingevoerde kleine letters binnen het programma zijn omgezet in hoofdletters. Als u het LIST-commando geeft, wordt deze wijziging automatisch uitgevoerd.

Functietoetsen

De functietoetsen bevatten de meest gebruikte commando's. De toetsen en de bijbehorende commando's zijn onder in het GW-BASIC-scherm aangegeven. U kunt veel sneller werken als u gebruik maakt van de toetsen.

Zo hoeft u voor het weergeven van een programma niet het commando LIST in te tikken. U hoeft alleen maar de betreffende functietoets in te drukken:

1 Druk op toets F1.

2 Druk op ENTER.

Vervolgens verschijnt het programma op het beeldscherm.

Om een programma te laten uitvoeren, hoeft u alleen maar op F2 te drukken. Onder deze toets is het commando RUN opgeslagen.

Hoofdstuk 4 geeft een overzicht van functietoetsen F3 tot en met F10.

Wijzigen van regels

U kunt regels wijzigen door:

- ze te wissen of te vervangen
- ze te wijzigen met het EDIT commando

U wist een regel door het regelnummer in te tikken en vervolgens op ENTER te drukken. Als u bijvoorbeeld 12 intikt en vervolgens op ENTER drukt, wordt regel 12 uit uw programma gewist.

Als u een regel wilt wijzigen, tik dan EDIT in en vervolgens het nummer van de regel die u wilt wijzigen.

Tik nu het volgende opdracht in en druk vervolgens op ENTER:

```
EDIT 10
```

Bij het wijzigen kunt u gebruik maken van de volgende toetsen:

Toets	Functie
CURSOR OMHOOG	verplaatst de cursor binnen de opdracht
CURSOR OMLAAG	verplaatst de cursor binnen de opdracht
CURSOR LINKS	verplaatst de cursor binnen de opdracht
CURSOR RECHTS	verplaatst de cursor binnen de opdracht
BACKSPACE	wist het teken links van de cursor
DELETE (DEL)	wist het teken onder de cursor
INSERT (INS)	laat u tekens invoeren aan de linkerkant van de cursor

Als u bijvoorbeeld regel 10 wilt veranderen in *X=4*, gebruik dan CURSOR RECHTS om de cursor op de 3 te plaatsen en tik vervolgens een 4 in. De 3 wordt dan vervangen door 4.

Druk nu op ENTER en vervolgens op F2.

Dan verschijnt op uw beeldscherm:

```
Ok
10 X=4
RUN
6
Ok
```

Opslaan van een programma

Het opstellen van een programma is in feite het aanmaken van een bestand. Een programma is niets meer dan een bestand met bepaalde instructies, of opdrachten, voor de computer. Als u een programma later nog eens wilt gebruiken, moet u het opslaan.

Voor het opslaan van een bestand in GW-BASIC gaat u als volgt te werk:

1 Druk op F4.

Het commando SAVE" verschijnt op het beeldscherm.

2 Tik een naam in voor het programma dat u wilt opslaan en druk op ENTER. Het bestand wordt dan opgeslagen met de door u opgegeven naam.

Als u een opgeslagen bestand wilt oproepen, gaat u als volgt te werk:

1 Druk op F3.

Het commando LOAD" verschijnt op het beeldscherm.

2 Tik de bestandsnaam in.

3 Druk op ENTER.

Het programma wordt geladen en u kunt het weergeven, wijzigen of uitvoeren.

4 DE SCREEN EDITOR VAN GW-BASIC

Programmaregels kunnen worden gewijzigd bij het invoeren of nadat ze zijn opgeslagen.

Regels wijzigen in nieuwe bestanden

U kunt een onjuist teken wissen met behulp van de toetsen BACKSPACE, DEL of CTRL-H. Na het wissen van een teken kunt u de regel afmaken.

Met ESC kunt u de regel waar u op dat moment mee bezig bent, wissen. Als u dus nog niet op ENTER hebt gedrukt en u wilt de huidige regel wissen, druk dan op ESC.

Als u een programma wilt wissen dat op dat moment in het geheugen is opgeslagen, tik dan NEW in. Dit commando wordt meestal gebruikt om het geheugen leeg te maken voordat u een nieuw programma invoert.

Regels wijzigen in opgeslagen bestanden

Soms wilt u veranderingen aanbrengen in uw GW-BASIC programma nadat u dit hebt opgeslagen. Geef hiertoe eerst het LIST-commando zodat het programma op het beeldscherm wordt weergegeven:

1 Laad het programma opnieuw.

2 Tik het LIST-commando in of druk op F1.

3 Geef het nummer, of de nummers, op van de regels die u wilt wijzigen.

De opgegeven regels verschijnen dan op uw beeldscherm.

WIJZIGEN VAN EEN PROGRAMMAREGEL

Zet de cursor op de plaats waar de wijzigingen moeten worden aangebracht. Wijzig daarna de inhoud op een van de volgende manieren:

- Schrijf over de bestaande tekens heen.
- Wis met behulp van BACKSPACE tekens links van de cursor.
- Wis tekens onder de cursor met behulp van DEL.
- Voeg tekens in vóór de cursor met behulp van de INS-toets. Het teken onder de cursor en alle volgende worden naar rechts geschoven en maken zo plaats voor het nieuwe teken.
- Voeg tekens toe aan het einde van de regel, of kort de regel in door aan het einde tekens te verwijderen.

Als u meerdere regels wijzigt, zorg dan wel dat u aan het einde van elke regel op ENTER drukt. De gewijzigde regels worden in de juiste numerieke volgorde opgeslagen, ook als u deze volgorde bij het wijzigen niet hebt aangehouden.

Opmerking:

De wijzigingen in een programmaregel worden pas in het GW-BASIC programma opgeslagen als u op de ENTER-toets drukt terwijl de cursor ergens in de gewijzigde regel staat.

Het is niet nodig de cursor aan het einde van de regel te plaatsen voordat u op ENTER drukt. De GW-BASIC-interpreter onthoudt namelijk waar een regel eindigt. Als u dus op ENTER drukt, wordt de hele regel opgeslagen, ook al staat de cursor aan het begin of in het midden van de regel.

Druk op CTRL-END of CTRL-E als u een regel op de huidige cursorpositie wilt afbreken. Druk daarna op ENTER.

Als u wilt dat de gewijzigde regel(s) worden opgeslagen, sla uw programma dan opnieuw op. Als u dit niet doet, worden de wijzigingen niet in het programma opgenomen.

Toetsen met speciale functies

De GW-BASIC-interpreter herkent negen van de numerieke toetsen aan de rechterkant van uw toetsenbord, de BACKSPACE-toets, de ESC-toets en de CTRL-toets. Hieronder volgt een opsomming van toetsen en combinaties van toetsen die binnen GW-BASIC speciale functies hebben:

Toets	Functie
BACKSPACE of CTRL-H	Wist het laatst ingetikte teken of het teken links van de cursor. De tekens rechts van de cursor worden een positie naar links verplaatst. Daaropvolgende tekens en schermregels binnen de huidige programmaregel worden verplaatst zoals bij de <DEL>-toets.
CTRL-BREAK of CTRL-C	Keert terug naar de commando-modus, zonder de wijzigingen op te slaan. Ook de automatische regelnummering (AUTO) wordt opgeheven.
CTRL-CURSOR-LINKS of CTRL-B	Verplaatst de cursor naar het begin van het vorige woord. Het vorige woord is het laatste alfanumerieke teken (A-Z en 0-9) links van de cursor.
CTRL-CURSOR-RECHTS of CTRL-F	Verplaatst de cursor naar het begin van het volgende woord. Het volgende woord is het volgende alfanumerieke teken rechts van de cursor. De cursor gaat dus naar het volgende cijfer of de volgende letter en slaat lege posities en speciale tekens over.
CURSOR-OMLAAG of CTRL -	Verplaatst de cursor een regel omlaag.
CURSOR-LINKS of CTRL]	Verplaatst de cursor een positie naar links. Als de cursor hierbij buiten de linkerkant van het beeldscherm wordt verplaatst, springt hij naar het begin van de volgende regel.
CURSOR-RECHTS of CTRL \	Verplaatst de cursor een positie naar rechts. Als de cursor hierbij buiten de rechterkant van het beeldscherm wordt verplaatst, springt hij naar het begin van de volgende regel.

CURSOR-OMHOOG of CTRL-6	Verplaatst de cursor een regel omhoog.
CTRL-BACKSPACE of DEL	Wist het teken onder de cursor. De tekens rechts van de cursor gaan een positie naar links en vullen zo de ruimte van het gewiste teken op. Als een regel meerdere regels van het beeldscherm in beslag neemt, verschuiven de tekens van de volgende regels een positie naar links. Het eerste teken van elke volgende regel verschuift naar het einde van de daaraan voorafgaande regel. De DEL-toets heeft de tegengestelde werking van de INS-toets. Als u tekst wist, wordt de regel korter.
CTRL-END of CTRL-E	Wist de tekst vanaf de cursor tot aan het einde van de opgegeven programmaregel. Dit houdt in dat alle tekst tot aan een <ENTER> wordt gewist.
CTRL-N of END	Verplaatst de cursor naar het einde van de opgegeven programmaregel. Tekens die u op deze positie intikt, worden toegevoegd.
CTRL-ENTER of CTRL-J	Verplaatst de cursor naar het begin van de volgende regel op het beeldscherm. Hierdoor kunt u programmaregels opstellen die langer zijn dan één schermregel. Een programmaregel mag maximaal 255 tekens lang zijn. U kunt deze functie ook gebruiken als "Line Feed".
CTRL-M of ENTER	Voegt een regel toe aan het GW-BASIC programma en verplaatst de cursor naar de volgende programmaregel.
CTRL [of ESC	Wist de hele programmaregel waarin de cursor op dat moment staat.
CTRL-G	Veroorzaakt een "beep"-geluid.
CTRL-K of HOME	Verplaatst de cursor naar de linkerbovenhoek van het beeldscherm. De inhoud van het beeldscherm wordt hierdoor niet gewijzigd.
CTRL-HOME CTRL-L	Maakt het beeldscherm leeg en plaatst de cursor in de linkerbovenhoek van het beeldscherm.
CTRL-R of INS	Schakelt de Insert-modus (insert = invoeren) in. De Insert-modus is ingeschakeld als de onderste helft van een tekenpositie is bedekt. In de grafische modus bedekt de cursor de gehele tekenpositie. Als de Insert-modus is uitgeschakeld, vervangen de ingetikte tekens de tekens die er al staan. Als u op de spatiebalk drukt, wordt het teken onder de cursor gewist en springt de cursor een positie naar rechts. De toets CURSOR-RECHTS verplaatst de cursor zonder het teken te wissen. Als de Insert-modus is uitgeschakeld en u op TAB drukt, springt de cursor naar de volgende tabulatorstop zonder dat de tekens worden verplaatst. Na elke acht tekenposities is een tabulatorstop aangebracht. Als de Insert-modus is ingeschakeld, worden bij het intikken van tekens de tekens achter de cursor naar rechts verplaatst. Steeds als een toets wordt ingedrukt, springt de cursor een positie naar rechts. Aan het einde van de regel wordt de tekst afgebroken en gaat verder op de volgende schermregel. Als u tekens invoegt, wordt de programmaregel langer.

	Als de Insert-modus is ingeschakeld en u op TAB drukt, worden vanaf de huidige positie tot aan de volgende tabulatorstop spaties toegevoegd.
CTRL-NUMLOCK of CTRL-S	Onderbreekt de uitvoering van een programma. Het programma wordt voortgezet zodra u op een toets drukt.
CTRL-PRTSC	Laat de scherminhoud direct op de printer afdrukken (LPT1:). Dit houdt in dat de door u ingetikte tekst direct wordt afgedrukt. Als u nogmaals op CTRL-PRTSC drukt, wordt deze functie uitgeschakeld.
SHIFT-PRTSC	Stuurt de scherminhoud naar de printer, zodat deze wordt afgedrukt.
CTRL-I of TAB	Verplaatst de cursor naar de volgende tabulatorstop. Na elke acht tekenposities is een tabulatorstop aangebracht.

Functietoetsen

Dankzij bepaalde toetsen of combinaties van toetsen kunt u veel commando's of functies uitvoeren met een minimum aantal toetshandelingen. De toetsen die dit mogelijk maken worden "functietoetsen" genoemd.

Als u bij het programmeren andere functies nodig hebt, kunt u de functies van deze toetsen aanpassen aan uw eigen behoeften door ze opnieuw te definiëren.

Onder elke functietoets kunt u maximaal 15 tekens opslaan, die u anders via 15 toetshandelingen zou moeten intikken. De toetsen hebben het opschrift F1 tot en met F10. GW-BASIC heeft enkele van de meest gebruikte functies toegekend aan deze toetsen. Deze functies verschijnen na het laden van GW-BASIC onderin het beeldscherm.

De volgende functies zijn aan de functietoetsen toegekend:

Functietoetsen van GW-BASIC

Toets	Functie	Toets	Functie
F1	LIST	F6	,"LPT1:"<-
F2	RUN<-	F7	TRON<-
F3	LOAD"	F8	TROFF<-
F4	SAVE"	F9	KEY
F5	CONT<-	F10	SCREEN 0,0,0<-

Opmerking:

Het symbool <- na een functie geeft aan dat u niet op ENTER hoeft te drukken na deze functie. Het commando wordt direct na het indrukken van de functietoets uitgevoerd.

U kunt de functietoetsen opnieuw definiëren (zie KEY en ON KEY).

5 AANMAKEN EN GEBRUIKEN VAN BESTANDEN

Een MS-DOS-systeem kent twee soorten bestanden:

- *Programmabestanden*, met het programma of de instructies voor de computer.
- *Gegevensbestanden*, met informatie die door programmabestanden wordt opgesteld of gebruikt.

Commando's voor programmabestanden

Hieronder volgt een overzicht van de meest gebruikte commando's en opdrachten voor programmabestanden (zie hoofdstuk 7).

SAVE *bestandsnaam*[,a][,p]

Het programma dat zich op dit moment in het geheugen bevindt, wordt op diskette opgeslagen.

LOAD *bestandsnaam*[,r]

Laadt het programma vanaf diskette in het geheugen. LOAD wist de huidige inhoud van het geheugen en sluit alle bestanden voordat het programma wordt geladen.

RUN *bestandsnaam*[,r]

Laadt het programma van diskette in het geheugen en laat het direct uitvoeren. RUN wist de huidige inhoud van het geheugen en sluit alle bestanden voordat het programma wordt geladen.

MERGE *bestandsnaam*

Laadt het programma vanaf diskette in het geheugen zonder het in het geheugen opgeslagen programma te wissen.

KILL *bestandsnaam*

Wist een diskette-bestand. Dit commando wordt ook gebruikt bij gegevensbestanden.

NAME *oude bestandsnaam* AS *nieuwe bestandsnaam*

Verandert de naam van een diskette-bestand. Alleen de naam van het bestand wordt gewijzigd. De inhoud blijft ongewijzigd en het bestand blijft op dezelfde positie op de diskette opgeslagen. Dit commando wordt ook gebruikt bij gegevensbestanden.

Gegevensbestanden

In een GW-BASIC programma kunnen twee soorten bestanden worden aangemaakt:

- Sequentiële bestanden.
- Onbepaald benaderbare bestanden.

Het aanmaken van sequentiële bestanden is eenvoudiger dan het aanmaken van onbepaald benaderbare bestanden. De flexibiliteit van sequentiële bestanden is echter kleiner en de toegangstijd is langer. In een sequentieel bestand worden de gegevens in de vorm van ASCII- codes opgeslagen. De gegevens worden achter elkaar (sequentieel) opgeslagen, in de volgorde waarin ze zijn ingevoerd en worden op dezelfde wijze ook weer ingelezen.

Voor het aanmaken en oproepen van een onbepaald benaderbaar bestand zijn meer stappen nodig dan voor sequentiële bestanden. Een onbepaald benaderbaar bestand neemt echter minder ruimte in beslag, omdat de gegevens in dergelijke bestanden als gecomprimeerde tekenreeksen worden opgeslagen.

In dit hoofdstuk wordt besproken hoe u deze twee soorten bestanden moet aanmaken en gebruiken.

AANMAKEN VAN EEN SEQUENTIEEL BESTAND

Bij sequentiële bestanden wordt gebruik gemaakt van de volgende opdrachten en functies:

CLOSE	LOF
EOF	OPEN
INPUT#	PRINT#
LINE INPUT#	PRINT# USING
LOC	UNLOCK
LOCK	WRITE#

Voor het aanmaken van een sequentieel bestand en het oproepen van de gegevens uit een dergelijk bestand gaat u als volgt te werk:

1 Open het bestand in de uitvoer (O) modus, zodat er gegevens naar dit bestand kunnen worden gestuurd. Daarna verschijnt het volgende op het beeldscherm:

```
OPEN "O",#1,"bestandsnaam"
```

2 Sla de gegevens op met behulp van de opdrachten PRINT# of WRITE#:

```
PRINT#1,A$
PRINT#1,B$
WRITE#1,C$
```

3 Voor het oproepen van de gegevens uit bestand, moet u het bestand eerst sluiten en opnieuw openen in de invoer (I) modus:

```
CLOSE #1
OPEN "I",#1,"bestandsnaam"
```

4 Lees de gegevens uit het sequentiële bestand in met behulp van INPUT# of LINE INPUT#:

```
INPUT#1,X$,Y$,Z$
```

Voorbeeld 1:
Dit voorbeeld geeft een kort programma waarmee het sequentiële bestand DATA wordt aangemaakt. De informatie is ingevoerd via het toetsenbord.

```
10 OPEN "O", #1, "DATA"
20 INPUT "Naam";N$
30 IF N$="stop" THEN END
40 INPUT "Afdeling";A$
50 INPUT "Sinds";H$
60 PRINT#1,N$;","A$","H$
70 PRINT:GOTO 20
RUN
Naam? Lex Goossen
Afdeling? Audio/Video
Sinds? 01/12/86

Naam? Helga van Strijp
Afdeling? Research
Sinds? 12/03/86

Naam? Peter Jacobs
Afdeling? Boekhouding
Sinds? 04/27/87

Naam? Ad Bruurs
Afdeling? Onderhoud
Sinds? 08/16/87

Naam? Voltooid
Ok
```

OPROEPEN VAN EEN SEQUENTIEEL BESTAND

Voorbeeld 2:
Het programma in voorbeeld 2 roept het in voorbeeld 1 aangemaakte bestand "DATA" aan en geeft de namen weer van de in 1987 aangestelde personen.

```
10 OPEN "I",#1,"DATA"
20 INPUT#1,N$,A$,H$
30 IF RIGHT$(H$,2)="87" THEN PRINT N$
40 GOTO 20
50 CLOSE #1
RUN
Peter Jacobs
Ad Bruurs
Bestandseinde gepasseerd in 20
Ok
```

Het programma in voorbeeld 2 leest de gegevens in het bestand in sequentiële volgorde. Als alle gegevens zijn gelezen, zorgt regel 20 voor de foutmelding "Bestandseinde gepasseerd". U kunt deze fout vermijden door regel 15 in te voegen. Deze regel gebruikt de EOF- functie, waarmee het einde van het bestand wordt gecontroleerd:

```
15 IF EOF (1) THEN END
```

Verander regel 40 in:

```
40 GOTO 15
```

Met een programma voor het aanmaken van een sequentieel bestand kunnen, met behulp van de PRINT# USING-opdracht, ook geformatteerde gegevens op diskette worden opgeslagen. Zo kunt u met behulp van de volgende opdracht numerieke gegevens op diskette opslaan, zonder zelf expliciet scheidingstekens op te geven:

```
PRINT#1,USING"####.##,";A,B,C,D
```

De komma achter de layout-tekenreeks dient als scheidingsteken tussen de gegevens in het bestand.

Als u in een sequentieel bestand de LOC-functie gebruikt, geeft deze het aantal records (elk 128 bytes) die sinds het openen van het bestand zijn opgeslagen of opgeroepen.

GEGEVENS TOEVOEGEN AAN EEN SEQUENTIEEL BESTAND

Bij het openen van een sequentieel bestand in de uitvoermodus wordt de huidige inhoud vernietigd. Als u gegevens wilt toevoegen aan een bestaand bestand zonder de inhoud ervan te vernietigen, moet u het bestand in de append-modus (A) openen.

Voorbeeld 3:
Met het programma in voorbeeld 3 kunt u een bestand *familie* aanmaken of aanvullen. In dit programma wordt het gebruik van LINE INPUT duidelijk. LINE INPUT accepteert alle ingetikte tekens (max. 255) en slaat deze op zodra op ENTER wordt gedrukt. Ook aanhalingstekens en komma's worden geaccepteerd.

```
100 OPEN "A",#1,"FAMILIE"
110 REM Voeg nieuwe gegevens in
120 INPUT "Naam"; N$
130 IF N$="" THEN 200 'Enter = stoppen
140 LINE INPUT "Adres? ";A$
150 LINE INPUT "Geb.datum? ";B$
160 PRINT#1,N$
170 PRINT#1,A$
180 PRINT#1,B$
190 PRINT:GOTO 120
200 CLOSE #1
```

Onbepaald benaderbare bestanden

In onbepaald benaderbare bestanden wordt informatie opgeslagen en opgeroepen in duidelijk genummerde eenheden, *records*. Aangezien de informatie genummerd is, kunnen gegevens worden opgeroepen vanaf een willekeurig punt van de diskette. In tegenstelling tot het zoeken naar sequentiële bestanden hoeft het programma niet de gehele diskette door te lezen om gegevens op te sporen. Met GW-BASIC kunnen grote onbepaald benaderbare bestanden worden aangemaakt. Het maximale aantal logische records is $2^{32}-1$.

Bij onbepaald benaderbare bestanden wordt gebruik gemaakt van de volgende opdrachten en functies:

CLOSE	GET	MKI$
CVD	LOC	MKS$
CVI	LOCK	OPEN
CVS	LOF	PUT
EOF	LSET/RSET	UNLOCK
FIELD	MKD$	

AANMAKEN VAN EEN ONBEPAALD BENADERBAAR BESTAND

Voor het aanmaken van een onbepaald benaderbaar gegevensbestand gaat u als volgt te werk:

1 Open het bestand in de onbepaald benaderbare (R) modus. In het onderstaande voorbeeld wordt een record-lengte van 32 bytes opgegeven. Als de record-lengte niet wordt opgegeven, wordt de standaardlengte van 128 bytes aangenomen.

```
OPEN "R",#1,"bestandsnaam",32
```

2 Reserveer ruimte in de bestandsbuffer voor de variabelen die in het onbepaald benaderbaar bestand worden opgeslagen. Gebruik hiervoor de FIELD-opdracht:

```
FIELD#1,20 AS N$,4 AS A$,8 AS P$
```

In dit voorbeeld worden de eerste 20 posities (bytes) van het onbepaald benaderbaar bestand gereserveerd voor de tekenreeks-variabele N$. De volgende 4 posities zijn gereserveerd voor A$. De daaropvolgende 8 posities voor P$.

3 Zorg met behulp van de opdrachten LSET en RSET dat de gegevens links, resp. rechts aangesloten in de velden worden opgeslagen.

Numerieke waarden moeten worden omgezet in tekenreeksen voordat ze in de buffer worden opgeslagen. MKI$ zet een geheel getal om in een tekenreeks. MKS$ zet getallen met enkele precisie om in tekenreeksen en MKD$ zet een getal met dubbele precisie om.

```
LSET N$ = X$
LSET A$ = MKS$(AMT)
LSET P$ = TEL$
```

4 Sla de gegevens uit de buffer op diskette op. Gebruik hiervoor de PUT-opdracht:

```
PUT#1,CODE%
```

Opmerking:
Gebruik in een INPUT- of LET-opdracht nooit een tekenreeks-variabele die in een FIELD-opdracht is gebruikt. Als een dergelijke variabele wel wordt gebruikt, verwijst de pointer van die variabele naar een tekenreeks-ruimte in plaats van naar ruimte in de bestandsbuffer.

Voorbeeld 4:
Het programma in voorbeeld 4 leest ingetikte gegevens in en slaat die vervolgens op in een onbepaald benaderbaar gegevensbestand. Steeds als de PUT-opdracht wordt uitgevoerd, wordt een record in het bestand opgeslagen. In dit voorbeeld wordt de CODE% van 2 cijferposities (regel 30) het record-nummer.

```
10 OPEN "R",#1"DEBITEUR",32
20 FIELD#1,20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "Klantcode (2 cijf.)";CODE%
40 INPUT "Naam";X$
50 INPUT "Bedrag";HOEV
60 INPUT "Telef."; TEL$:PRINT
70 LSET N$=X$
80 LSET A$=MKS$(HOEV)
90 LSET P$=TEL$
100 PUT #1,CODE%
110 GOTO 30
```

OPROEPEN VAN EEN ONBEPAALD BENADERBAAR BESTAND

Voor het oproepen van een onbepaald benaderbaar bestand gaat u als volgt te werk:

1 Open het bestand in de R-modus:

OPEN "R",#1,"*bestandsnaam*",32

2 Reserveer ruimte in de bestandsbuffer voor de variabelen die u wilt oproepen. Gebruik hiervoor de FIELD-opdracht:

FIELD#1,20 AS N$, 4 AS A$, 8 AS P$

In dit voorbeeld worden de eerste 20 posities van het onbepaald benaderbaar bestand gereserveerd voor de tekenreeks-variabele N$. De volgende 4 posities zijn gereserveerd voor A$. De daaropvolgende 8 posities voor P$.

Opmerking:
In een programma waarin een INPUT- en een OUTPUT-opdracht voor hetzelfde bestand wordt gebruikt, kan vaak slechts één OPEN- opdracht en één FIELD-opdracht worden gebruikt.

3 Sla het gewenste record op in de bestandsbuffer met behulp van de GET-opdracht:

GET #1,CODE%

De gegevens in het bestandsbuffer kunnen nu door het programma worden opgeroepen.

4 Zet de numerieke waarden weer om in getallen. Gebruik hiervoor de volgende conversiefuncties: CVI voor gehele getallen, CVS voor getallen met enkele precisie en CVD voor getallen met dubbele precisie.

```
PRINT N$
PRINT CVS (A$)
.
.
.
```

Voorbeeld 5:
Met het programma in voorbeeld 5 wordt het in voorbeeld 4 opgestelde onbepaald benaderbaar bestand opgeroepen. Door een code van 2 cijferposities op te geven, wordt de bij die code behorende informatie uit het bestand opgeroepen en weergegeven.

```
10 OPEN "R",#1,"DEBITEUR",32
20 FIELD#1, 20 AS N$, 4 AS A$, 8 AS P$
30 INPUT "Klantcode (2 cijf.)";CODE%
40 GET #1, CODE%
50 PRINT N$
60 PRINT USING "$$###.##";CVS(A$)
70 PRINT P$:PRINT
80 GOTO 30
```

Bij onbepaald benaderbare bestanden geeft LOC het huidige record-nummer aan. Het huidige record-nummer komt overeen met het laatst gebruikte record-nummer in een GET of PUT opdracht. Zo beëindigt de volgende programmaregel de uitvoering van het programma als het huidige record-nummer in bestand#1 hoger is dan 99:

```
IF LOC(1)#99 THEN END
```

Voorbeeld 6:
Dit voorbeeld verduidelijkt het oproepen van een onbepaald benaderbaar bestand. In dit programma wordt het record-nummer gebruikt als onderdeelnummer. Het bevat maximaal 100 verschillende onderdeelnummers.

In de regels 900-960 wordt het gegevensbestand geïnitialiseerd door CHR$(255) op te geven als eerste teken voor elk record. Met behulp van dit teken wordt later (regel 270 en regel 500) nagegaan of er al een onderdeel met dat nummer bestaat.

De regels 130-220 geven de verschillende functies van het programma weer. Als u het gewenste functienummer hebt opgegeven, springt het programma op regel 230 naar de bijbehorende subroutine.

```
120 OPEN"R",#1,"VOORRAAD.DAT",39
125 FIELD#1,1 AS F$, 30 AS D$, 2 AS Q$, 2 AS R$, 4 AS
    P$
130 PRINT:PRINT "Functies:":PRINT
135 PRINT 1,"Initialiseer bestand"
140 PRINT 2,"Nieuw onderdeel invoeren"
150 PRINT 3,"Geef voorraad van een onderdeel"
160 PRINT 4,"Toevoegen aan voorraad"
170 PRINT 5,"Aftrekken van voorraad"
180 PRINT 6,"Geef onderdelen waar tekort aan is"
220 PRINT:PRINT:INPUT"Wat is uw keuze";KEUZE
225 IF (KEUZE<1) OR (KEUZE>6) THEN PRINT "Onjuist
    functienummer":GOTO 130
230 ON KEUZE GOSUB 900, 250, 390, 480, 560, 680
240 GOTO 220
250 REM Nieuw onderdeel invoeren
260 GOSUB 840
270 IF ASC(F$)<>255 THEN INPUT"Bestaat al; over
    schrijven";A$:IF A$<>"Y" THEN RETURN
280 LSET F$=CHR$(0)
290 INPUT "Beschrijving";DESC$
300 LSET D$=DESC$
310 INPUT "Hoev. op voorraad";Q%
320 LSET Q$=MKI$(Q%)
330 INPUT "Minimaal vereiste voorraad";R%
340 LSET R$=MKI$(R%)
350 INPUT "Stukprijs";P
360 LSET P$=MKS$(P)
370 PUT#1,OND%
380 RETURN
```

```
390 REM Geef voorraad van een onderdeel
400 GOSUB 840
410 IF ASC(F$)=255 THEN PRINT "Onderdeel bestaat
    niet":RETURN
420 PRINT USING "Ond.nummer ###";OND%
430 PRINT D$
440 PRINT USING "Aanw. voorraad #####";CVI(Q$)
450 PRINT USING "Minimaal vereiste voorraad
    ######";CVI(R$)
460 PRINT USING "Stukprijs $$##.##";CVS(P$)
470 RETURN
480 REM Toevoegen aan voorraad
490 GOSUB 840
500 IF ASC(F$)=255 THEN PRINT "Onderdeel bestaat
    niet":RETURN
510 PRINT D$:INPUT "Toe te voegen hoev.";A%
520 Q%=CVI(Q$)+A%
530 LSET Q$=MKI$(Q%)
540 PUT#1, OND%
550 RETURN
560 REM Aftrekken van voorraad
570 GOSUB 840
580 IF ASC(F$)=255 THEN PRINT "Onderdeel bestaat
    niet":RETURN
590 PRINT D$
600 INPUT "Af te trekken hoev.":S%
610 Q%=CVI(Q$)
620 IF (Q%-S%)<0 THEN PRINT "Slechts";Q%;"op
    voorraad":GOTO 600
630 Q%=Q%-S%
640 IF Q%<=CVI(R$) THEN PRINT "Voorraad nu";Q%;";
    minimaal vereist is:";CVI(R$)
650 LSET Q$=MKI$(Q%)
660 PUT#1, OND%
670 RETURN
680 REM Geef onderdelen waar tekort aan is
690 FOR I=1 TO 100
710 GET#1,I
720 IF CVI(Q$)<CVI(R$) THEN PRINT D$;"Hoev.";CVI(Q$)
    TAB(50) "Minimaal vereist:";CVI(R$)
730 NEXT I
740 RETURN
840 INPUT "Onderdeelnummer";OND%
850 IF (OND%<1) OR (OND%>100) THEN PRINT "Onjuist
    ond.nummer": GOTO 840 ELSE GET#1,OND%:RETURN
890 END
900 REM Initialiseer bestand
910 INPUT "Weet u het zeker";B$:IF B$ <> "y" THEN
    RETURN
920 LSET F$=CHR$(255)
930 FOR I=1 TO 100
940 PUT#1,I
950 NEXT I
960 RETURN
```

6 CONSTANTEN, VARIABELEN, EXPRESSIES EN OPERATOREN

Nu u op de hoogte bent van de belangrijkste aspecten van het programmeren in GW-BASIC, kunnen we overstappen op het schrijven van langere programma's. Dit hoofdstuk gaat nader in op het gebruik van constanten, variabelen, expressies en operatoren in GW-BASIC en geeft aan hoe u deze kunt gebruiken voor het opstellen van complexere programma's.

Constanten

Constanten zijn variabelen waarvan de inhoud steeds hetzelfde blijft. Er zijn twee soorten constanten: *tekenreeks-constanten* en *numerieke constanten.*

Een *tekenreeks-constante* is een combinatie van 0 tot 255 alfanumerieke tekens tussen aanhalingstekens. Bijvoorbeeld:

"HALLO"
"ƒ 25.000,00"
"Aantal employees"

Numerieke constanten kunnen positief of negatief zijn. Gebruik bij het intikken van numerieke constanten in GW-BASIC geen komma's om duizendtallen aan te geven. Als u bijvoorbeeld het getal 10,250 wilt invoeren, moet u het intikken als 10250. GW-BASIC beschouwt een komma namelijk als scheidingsteken. Zou u een komma gebruiken, dan ziet GW-BASIC 10,250 dus als twee getallen, namelijk 10 en 250. Er bestaan vijf soorten numerieke constanten: *geheel getal* (*integer*), *getal met decimale punt*, *decimaal getal*, *hexadecimaal getal* en *octaal getal.*

Constante	Beschrijving
Geheel getal	Gehele getallen tussen -32768 en +32767. Deze getallen bevatten geen decimale punt.
Getal met decimale punt	Positieve of negatieve getallen met een decimale punt.
Decimaal getal	Positieve of negatieve getallen weergegeven in exponentiële notatie (ook wel eens "wetenschappelijke notatie" genoemd). Een decimaal getal bestaat uit een geheel getal of een getal met een decimale punt (de mantisse), gevolgd door de letter E en een geheel getal (de exponent) dat soms wordt voorafgegaan door een plus- of minteken. De exponentiële notatie schrijft zeer grote of zeer kleine getallen als een macht van 10. Bijvoorbeeld: $1000=10^3$, zou in exponentiële notatie worden geschreven als 10E+3.

	Decimale getallen variëren tussen 3.0 x 10^{-39} tot 1.7 x 10^{38}. Bijvoorbeeld: 235.988E-7 = .0000235988 2359E6 = 2359000000
Hexadecimaal getal	Hexadecimale getallen voorafgegaan door &H. Bijvoorbeeld: &H76 &H32F
Octaal getal	Octale getallen voorafgegaan door & of &O. Bijvoorbeeld: &O357 &1234

NUMERIEKE CONSTANTEN MET ENKELE PRECISIE EN MET DUBBELE PRECISIE

Numerieke constanten kunnen gehele getallen, getallen met dubbele precisie of getallen met enkele precisie zijn. Bij gehele getallen worden nooit decimalen opgeslagen. Getallen met enkele precisie worden opgeslagen met 7 cijferposities (hoewel met slechts 6 posities nauwkeurig wordt gerekend). Getallen met dubbele precisie worden opgeslagen met 17 cijferposities, en worden afgedrukt met 16 cijferposities.

Een constante met enkele precisie heeft een of meer van de volgende kenmerken:

- bevat 7 cijferposities of minder
- exponentiële vorm met E
- wordt gevolgd door een uitroepteken (!)

Een constante met dubbele precisie heeft een of meer van de volgende kenmerken:

- bevat 8 cijferposities of meer
- exponentiële vorm met D
- wordt gevolgd door het hekje (#)

Voorbeelden van numerieke constanten met enkele precisie en dubbele precisie:

Constanten met enkele precisie	Constanten met dubbele precisie
46.8	345692811
-1.09E-06	-1.09432D-06
3498.0	3490.0#
22.5!	7654321.1234

Variabelen

Variabelen dienen voor het opslaan en weergeven van waarden. De waarde van een variabele wordt opgegeven, of is het resultaat van berekeningen in het programma. Als voor een variabele geen waarde wordt opgegeven, wordt aangenomen dat de waarde nul is.

NAMEN EN TYPE-AANDUIDINGEN VAN VARIABELEN

De maximale lengte van variabelenamen in GW-BASIC is 40. U kunt langere variabelenamen gebruiken, maar de resterende tekens worden door GW-BASIC genegeerd. In een variabele mag gebruik worden gemaakt van letters, cijfers en de decimale punt. Het eerste teken moet een letter zijn. Ook mogen speciale type-aanduidingen worden gebruikt.

Woorden die worden gebruikt als commando's, opdrachten, functies en operatoren kunnen niet worden gebruikt als variabelenaam. Het is echter wel toegestaan een gereserveerd woord als deel van een variabelenaam te gebruiken. Voorbeeld: de variabelenaam RUN is niet toegestaan, maar RUNDEREN wel.

In variabelen kunnen zowel numerieke waarden als tekenreeksen worden opgeslagen.

TEKENS VOOR TYPE-AANDUIDINGEN

Voor het aanduiden van het type van de variabele worden de volgende tekens gebruikt:

Tekentype	Variabele
$	tekenreeks-variabele
%	geheel getal (integer)
!	variabele met enkele precisie
#	variabele met dubbele precisie

Voorbeelden van namen voor elk type variabele:

Type	Voorbeeld-naam
Tekenreeks	NAAM$, adres$
Geheel getal	Getal%, aantal%
Enkele precisie	MINIMUM!
Dubbele precisie	P1#

Als geen type wordt opgegeven, wordt de variabele gezien als een variabele met enkele precisie. Met variabelen met dubbele precisie kunnen berekeningen nauwkeuriger worden uitgevoerd, maar ze vereisen meer geheugenruimte en rekentijd. Voor de meeste toepassingen is de nauwkeurigheid enkele precisie voldoende, alhoewel het zevende cijfer (als dit wordt weergegeven) niet altijd even nauwkeurig is. Ga altijd zeer voorzichtig te werk bij berekeningen waarin combinaties van gehele getallen, getallen met enkele precisie en getallen met dubbele precisie worden gebruikt.

De volgende variabele wordt door het systeem gezien als een getal met enkele precisie, omdat u zelf geen type-aanduiding hebt opgegeven:

ABC

Variabelen waarvan de naam begint met FN zijn door de gebruiker zelf gedefinieerde functies.

De opdrachten DEFINT, DEFSTR, DEFSNG en DEFDBL kunnen in een programma worden opgenomen om een type toe te kennen aan variabelen waarvan de naam begint met een opgegeven letter.

MATRIX VARIABELEN

Een matrix is een groep of een tabel met waarden waarnaar via één variabelenaam wordt verwezen. Naar elke cel binnen een matrix wordt verwezen door een aan de variabelenaam toegevoegd getal. Dit bestaat uit een geheel getal tussen haakjes, dat verwijst naar het celnummer binnen de *matrix-dimensie*. Het aantal getallen achter de matrix-naam is even groot als het aantal dimensies in een matrix.

Bijvoorbeeld:

```
V(10)
```

verwijst naar een waarde in een één-dimensionale matrix

```
T(1,4)
```

verwijst naar een waarde in een twee-dimensionale matrix.

Het maximale aantal dimensies voor een matrix in GW-BASIC is 255. Het maximale aantal cellen per dimensie is 32767.

Opmerking:

Zodra u een matrix gebruikt met een dimensie waarvan de waarde groter is dan 10, dan moet u de DIM-opdracht gebruiken (zie de beschrijving van DIM in hoofdstuk 7). Als u een cel opvraagt waarvan het nummer groter is dan de hoogste dimensie-waarde, dan verschijnt de foutmelding "Te hoog indexgetal".

Meerdimensionale matrices (meerdere dimensies, gescheiden door komma's) zijn erg geschikt voor het opslaan van tabellen. Zo kan A(1,4) worden gebruikt voor het weergeven van een matrix van twee rijen en vijf kolommen:

Kolom	0	1	2	3	4
Rij 0	10	20	30	40	50
Rij 1	60	70	80	90	100

In dit voorbeeld geldt: cel A(1,2)=80 en A(0,3)=40

Tenzij dit anders is aangegeven, beginnen rijen en kolommen altijd met 0 en niet met 1 (zie OPTION BASE in hoofdstuk 7).

VEREISTE GEHEUGENRUIMTE VOOR HET OPSLAAN VAN VARIABELEN

Elk type variabele neemt een andere hoeveelheid geheugenruimte in beslag. In verband met de opslag- en geheugencapaciteit van uw computer en met het programma dat u opstelt, zijn de onderstaande getallen voor u van belang.

Variabele	Aantal bytes
Geheel getal	2
Enkele precisie	4
Dubbele precisie	8

Matrix	Aantal bytes
Geheel getal	2 per cel
Enkele precisie	4 per cel
Dubbele precisie	8 per cel

Tekenreeksen:

Drie bytes extra, plus een byte voor elk teken van de tekenreeks. Aanhalingstekens die het begin en het einde van de tekenreeks aanduiden, nemen geen ruimte in beslag.

Type conversie

Indien nodig, zet GW-BASIC het type van een numerieke variabele om in een ander type. Hierbij worden de volgende regels in acht genomen:

- Als een numeriek variabele van een bepaald type wordt gelijkgesteld aan een numerieke variabele met een ander type, wordt het getal opgeslagen met het type dat in de variabelenaam is aangegeven. Bijvoorbeeld:

```
10 A% = 23.42
20 PRINT A%
RUN
 23
```

  Als u probeert een tekenreeks op te slaan in een numerieke variabele of probeert een getal op te slaan in een tekenreeks-variabele, dan verschijnt de foutmelding "Onjuist type".

- Tijdens het uitvoeren van een rekenkundige bewerking worden alle numerieke variabelen in die bewerking omgezet in dezelfde precisie, namelijk de precisie van het meest nauwkeurige variabele-type. Hierdoor wordt ook het resultaat van deze bewerking omgezet in deze precisie. Bijvoorbeeld:

```
10 D# = 6#/7
20 PRINT D#
RUN
 .8571428571428571
```

  De berekening wordt uitgevoerd in dubbele precisie en ook het resultaat D# wordt weergegeven als een getal met dubbele precisie.

```
10 D = 6#/7
20 PRINT D
RUN
.8571429
```

  De berekening wordt uitgevoerd in dubbele precisie, maar het resultaat D wordt nu weergegeven als een getal met enkele precisie.

- Logische operatoren zetten getallen om in gehele getallen en geven als resultaat een geheel getal weer. De getallen moeten in het bereik -32768 t/m 32767 liggen. Als u een getal buiten dit bereik opgeeft, volgt de foutmelding "Te groot getal".

- Bij het converteren van een decimaal getal naar een geheel getal, wordt het getal afgerond. Bijvoorbeeld:

```
10 C% = 55.88
20 PRINT C%
RUN
 56
```

- Als in een variabele met dubbele precisie een getal met enkele precisie wordt opgeslagen, dan wordt alleen met de eerste zeven cijfers van dat getal (afgerond) gerekend. Een getal met enkele precisie heeft namelijk slechts een nauwkeurigheid van zeven cijfers. De absolute waarde van het verschil tussen het met dubbele precisie weergegeven getal en het originele getal met enkele precisie is minder dan 6.3E-8 maal het originele getal met enkele precisie. Bijvoorbeeld:

```
10 A = 2.04
20 B# = A
30 PRINT A;B#
RUN
2.04 2.039999961853027
```

Expressies en operatoren

Een expressie is een tekenreeks-constante, numerieke constante, een variabele of een combinatie van constanten en variabelen, waarmee door het tussenvoegen van operatoren één uitkomst wordt verkregen.

Operatoren worden gebruikt voor het uitvoeren van wiskundige en logische bewerkingen. GW-BASIC kent vier soorten operatoren:

- Wiskundige
- Relationele
- Logische
- Functionele

WISKUNDIGE OPERATOREN

GW-BASIC maakt gebruik van de onderstaande wiskundige operatoren. Ze worden gegeven in volgorde waarin zij worden uitgevoerd:

Operator	Bewerking
^	Machtsverheffen
-	Negatief maken
*	Vermenigvuldigen
/	Delen
+	Optellen
-	Aftrekken

Bewerkingen tussen haakjes worden eerst uitgevoerd. Binnen de haakjes wordt de gebruikelijke volgorde van bewerking aangehouden.

Hierna volgt een aantal voorbeelden van algebraïsche expressies en hun equivalenten in GW-BASIC:

Algebraïsche expressie	BASIC expressie
$\frac{X-Y}{Z}$	(X-Y)/Z
$\frac{XY}{Z}$	X*Y/Z
$\frac{X+Y}{Z}$	(X+Y)/Z
$(X^2)^Y$	(X^2)^Y
X^{Y^Z}	X^(Y^Z)
X(-Y)	X*(-Y)

Twee opeenvolgende operatoren moeten altijd worden gescheiden door haakjes.

Integer-delen en modulo

Naast de hierboven genoemde bewerkingen, zijn nog twee rekenkundige bewerkingen mogelijk: integer-delen en modulo.

De schuine streep (\) is de operator voor integer-delen. De getallen in een dergelijke berekening worden afgerond tot gehele getallen (moet een getal zijn tussen -32768 en 32767) voordat de deling wordt uitgevoerd. De uitkomst wordt ingekort tot een geheel getal.

Voorbeelden van integer-delen:

```
13\4 = 3

25.68\6.99 = 3
```

Integer-delen komt in de bewerkingsvolgorde na het delen met decimale getallen.

Een modulo wordt aangegeven door de operator MOD. Deze bewerking geeft de rest van een integer-delen.

Voorbeeld van een modulo:

```
13 MOD 4 = 1
(13 gedeeld door 4 = 3 rest 1)

25.68 MOD 6.99 = 5
(26 gedeeld door 7 = 3 rest 5)
```

De modulo komt in de volgorde van bewerking na integer-delen. Bij modulo-bewerkingen zijn ook de functies INT en FIX zeer nuttig (zie hoofdstuk 7).

Te groot getal en delen door nul

Als in een bewerking wordt gedeeld door nul, verschijnt de foutmelding "Deling door nul". Het grootste getal dat het systeem kan weergeven wordt als resultaat van die bewerking gegeven en vervolgens wordt de bewerking verder uitgevoerd.

Als het getal 0 tot een negatieve macht wordt verheven, volgt de foutmelding "Deling door nul". Het grootste getal dat het systeem kan weergeven wordt als resultaat van die bewerking gegeven en vervolgens wordt de bewerking verder uitgevoerd.

Bij een te groot getal verschijnt de foutmelding "Te groot getal", Het grootste getal dat het systeem kan weergeven, inclusief plus- of minteken, wordt als resultaat van die bewerking gegeven en vervolgens wordt de bewerking verder uitgevoerd. Te grote getallen en delen door nul worden niet opgespoord door de functies voor foutdetectie.

RELATIONELE OPERATOREN

Met behulp van relationele operatoren kunt u twee waarden vergelijken. Het resultaat van een dergelijke vergelijking is of waar (true, -1) of onwaar (false, 0). Dit resultaat kan worden gebruikt om een beslissing te nemen omtrent de voortzetting van het programma.

In de onderstaande tabel worden de relationele operatoren weergegeven.

Relationele operatoren

Operator	Geteste relatie	Expressie
=	Gelijk aan	X = Y
<>	Ongelijk aan	X <> Y
<	Kleiner dan	X < Y
>	Groter dan	X > Y
<=	Kleiner dan of gelijk aan	X <= Y
>=	Groter dan of gelijk aan	X >= Y

Het gelijk-teken wordt ook gebruikt voor het toekennen van een waarde aan een variabele (zie LET-opdracht).

Rekenkundige en relationele operatoren kunnen samen in een expressie worden gebruikt. De rekenkundige bewerking wordt altijd eerst uitgevoerd.

```
X+Y < (T-1)/Z
```

Deze vergelijking is waar als de waarde van X plus Y kleiner is dan de waarde van T-1 gedeeld door Z.

LOGISCHE OPERATOREN

Logische operatoren worden gebruikt voor bit-manipulaties, Booleaanse bewerkingen of het testen van meervoudige voorwaarden. Het resultaat van een logische bewerking wordt weergegeven in bits met de status waar (true, ongelijk aan 0) of onwaar (false, gelijk aan 0). In een expressie worden de logische bewerkingen uitgevoerd na rekenkundige bewerkingen en relationele bewerkingen. In de onderstaande tabel is te zien hoe de resultaten van een logische bewerking worden bepaald. De operatoren zijn weergegeven in volgorde van bewerking. In deze tabel duidt T op true (waar, ongelijk aan 0) en F op false (onwaar, gelijk aan 0).

Resultaten van een logische bewerking

Bewerking	Waarde	Waarde	Resultaat
NOT	X		NOT X
	T		F
	F		T
AND	X	Y	X AND Y
	T	T	T
	T	F	F
	F	T	F
	F	F	F
OR	X	Y	X OR Y
	T	T	T
	T	F	T
	F	T	T
	F	F	F
XOR	X	Y	X XOR Y
	T	T	F
	T	F	T
	F	T	T
	F	F	F
EQV	X	Y	X EQV Y
	T	T	T
	T	F	F
	F	T	F
	F	F	T
IMP	X	Y	X IMP Y
	T	T	T
	T	F	F
	F	T	T
	F	F	T

Zoals relationele operatoren kunnen worden gebruikt voor het nemen van beslissingen op grond van gestelde voorwaarden, zo kunnen logische operatoren worden gebruikt voor het koppelen van twee of meer voorwaarden en voor het nemen van beslissingen op basis van de toestanden "waar" of "onwaar". Bijvoorbeeld:

```
IF D<200 AND F<4 THEN 80
IF I>10 OR K<0 THEN 50
IF NOT P THEN 100
```

Logische operatoren zetten hun operand om in 16-bit gehele getallen, waarop de "two's complement" bewerking is uitgevoerd en die zijn voorzien van een teken. De waarde van deze gehele getallen moet liggen tussen -32768 en +32767. Als beide operands 0 of -1 zijn, is het resultaat van een logische berekening ook 0 of -1. De bewerkingen op deze gehele getallen worden per afzonderlijk bit uitgevoerd. Dit houdt in dat elk bit in het resultaat wordt bepaald door de status van de twee daarmee in de operands corresponderende bits.

Het is dus mogelijk met behulp van logische operatoren bytes te testen voor een bepaald bit-patroon. Zo kan de AND-operator worden gebruikt om één bit uit een bepaald bit-patroon te lichten. De OR-operator kan worden gebruikt voor het samenvoegen van twee bytes om een bepaald bit-patroon te realiseren. De onderstaande voorbeelden tonen de werking van logische operatoren:

Voorbeeld	Uitleg
63 AND 16=16	63=binair 111111 en 16=binair 10000. dus 63 AND 16=16
15 AND 14=14	15=binair 1111 en 14=binair 1110, dus 15 AND 14=14 (binair 1110)
-1 AND 8=8	-1=binair 1111111111111111 en 8=binair 1000, dus -1 AND 8=8
4 OR 2=6	4=binair 100 en 2=binair 10, dus 4 OR 2=6 (binair 110)
10 OR 10=10	10=binair 1010, dus 1010 OR 1010=1010 (10)
-1 OR-2=-1	-1=binair 1111111111111111 en -2=binair 1111111111111110, dus -1 OR -2=-1. De "bit-complement" van 16 nullen is 16 enen. Dit is de "two's complement" van -1.
NOT X=-(X+1)	De "two's complement" van een geheel getal is de "bit complement" plus één.

FUNCTIONELE OPERATOREN

Een functie wordt gebruikt in een expressie voor het aanroepen van een voorgeprogrammeerde bewerking. GW-BASIC heeft in het systeem opgeslagen functies, zoals SQR (vierkantswortel) of SIN (sinus).

GW-BASIC accepteert ook door de gebruiker opgestelde functies (zie DEF FN-opdracht).

TEKENREEKS-OPERATOREN

Voor het vergelijken van tekenreeksen worden dezelfde operatoren gebruikt als bij het vergelijken van getallen:

Operator	Betekenis
=	Gelijk aan
<>	Ongelijk aan
<	Kleiner dan
>	Groter dan
<=	Kleiner dan of gelijk aan
>=	Groter dan of gelijk aan

De GW-BASIC-interpreter vergelijkt tekenreeksen door steeds één teken uit elke tekenreeks te vergelijken met een teken uit de andere tekenreeks, op basis van de ASCII-codes. Als de ASCII-codes van de tekenreeksen overeenkomen, zijn de tekenreeksen gelijk. Als de ASCII-codes niet gelijk zijn, is de tekenreeks met de laagste ASCII-code "kleiner dan" de andere tekenreeks. Als de interpreter bij het vergelijken van tekenreeksen het einde van een van deze tekenreeksen bereikt, dan beschouwt GW-BASIC de kortere tekenreeks als "kleiner dan" de langere, verondersteld dat beide tekenreeksen tot dat moment aan elkaar gelijk zijn. Spaties aan het begin en het einde van een tekenreeks hebben betekenis.

Bijvoorbeeld:

```
"AA" < "AB"
"BESTANDSNAAM" = "BESTANDSNAAM"
"X&" > "X#"
"CL " > "CL"
"kg" > "KG"
"JAN" < "JANTJE"
B$ < "9/12/87", waarbij B$ = "8/12/87"
```

Tekenreeks-vergelijkingen kunnen ook worden gebruikt voor het testen van tekenreeks-waarden of voor het alfabetiseren van tekenreeksen. Tekenreeks-constanten die worden gebruikt in vergelijkingen moeten tussen haakjes gezet worden.

Tekenreeksen kunnen een elkaar worden verbonden door het plusteken (+). Bijvoorbeeld:

```
10 A$="Bestands": B$="naam"
20 PRINT A$+B$
30 PRINT "Nieuwe" + A$+B$
RUN
Bestandsnaam
Nieuwe Bestandsnaam
Ok
```

7 COMMANDO'S, OPDRACHTEN EN FUNCTIES

Dit deel van de handleiding geeft een alfabetisch gerangschikt overzicht van alle instructies in GW-BASIC: opdrachten, functies, commando's en variabelen.

Elke beschrijving begint met de naam en het type van de instructie en daarna volgen:

Functie:	De bewerking die door de instructie wordt uitgevoerd.
Syntaxis:	De juiste notatiewijze van de instructie.
Opmerkingen:	Aanvullende informatie over de instructie en de effecten bij het uitvoeren.
Voorbeelden:	Geeft aan hoe de instructie in een programma kan worden gebruikt.
Attentie:	Speciale voor de betreffende instructie belangrijke informatie.

ABS functie

Functie:
Berekening van de absolute waarde van een getal.

Syntaxis:
ABS(n)

Opmerkingen:
n moet een numerieke expressie zijn.

Voorbeeld:

```
PRINT ABS(7*(-5))
35
Ok
```

Op het scherm verschijnt 35, de uitkomst van de berekening.

ASC functie

Functie:
Bepaling van de ASCII-code voor het eerste teken in een tekenreeks.

Syntaxis:
ASC(x$)

Opmerkingen:
Als x$ leeg is (dus een "nul-string" is), dan volgt de foutmelding "Onjuiste functieaanroep".
Als x$ met een hoofdletter begint, dan geeft ASC(x$) een waarde tussen 65 en 90.
Als x$ met een kleine letter begint, dan verschijnt een waarde tussen 97 en 122.

De getallen 0 t/m 9 hebben de ASCII-codes 48 t/m 57.

Raadpleeg ook de CHR$-functie voor conversie van ASCII-codes naar alfanumerieke tekens.

Appendix C geeft een overzicht van de ASCII-codes in GW-BASIC.

Voorbeeld:

```
10 X$="Telefoon"
20 PRINT ASC(X$)
30 RUN
 84
Ok
```

De ASCII-code voor de letter *T* is 84.

ATN functie

Functie:
Berekening van de arctangens voor een hoek in radialen.

Syntaxis:
ATN(x)

Opmerkingen:
De uitkomst is een getal tussen -pi/2 en pi/2.

Expressie *x* kan elk type numerieke waarde zijn. De berekening wordt in enkele precisie uitgevoerd, behalve wanneer via de optie /d de dubbele precisie is geactiveerd.

U kunt graden omzetten in radialen door te vermenigvuldigen met pi/180.

Voorbeeld:

```
10 INPUT X
20 PRINT ATN(X)
RUN
? 3
1.249046
Ok
```

De arctangens voor een hoek van 3 radialen is 1.249046.

AUTO commando

Functie:
Automatisch genereren van regelnummers voor programmaregels.

Syntaxis:
AUTO [*regelnummer*[,[*interval*]]
AUTO .[,[*interval*]]

Opmerkingen:
AUTO is zeer nuttig bij het invoeren van programma's omdat de gebruiker de regelnummers niet zelf hoeft in te tikken.

Het eerste regelnummer komt overeen met *regelnummer*. Elk volgend regelnummer is steeds een *interval* hoger. De standaardwaarde voor beide argumenten is 10.

Als u *regelnummer* vervangt door een punt (.), dan begint het effect van AUTO bij het huidige regelnummer.

Als u *regelnummer* vervangt door een komma en geen *interval* opgeeft, dan hanteert AUTO het *interval* van het laatstgebruikte AUTO-commando.

Als via AUTO een bestaand regelnummer wordt gegenereerd, dan staat na dat regelnummer een sterretje (*). Dit teken waarschuwt dat de daarna ingetikte programmaregel de bestaande regel met hetzelfde regelnummer zal vervangen. Drukt u echter alleen op ENTER, dan blijft de bestaande regel intact en verschijnt het eerstvolgende regelnummer. U heft de AUTO-bewerking op via CTRL-BREAK of CTRL-C. GW-BASIC keert dan terug naar de commando-modus.

Attentie:
De regel met de opdracht CTRL-BREAK of CTRL-C wordt niet opgeslagen. Om te garanderen dat u alleen de gewenste opdrachten opslaat, is het raadzaam de opdrachten CTRL-BREAK of CTRL-C niet te gebruiken op regels met alleen een regelnummer.

Voorbeelden:
```
AUTO 100,50
```

Genereert de regelnummers 100, 150, 200, 250, enz.

```
AUTO
```

Genereert de regelnummers 10, 20, 30, 40, enz., met de standaard interval-waarde 10 omdat geen interval werd opgegeven.

BEEP opdracht

Functie:
Genereren van een pieptoon van 800 Hz (800 trillingen per seconde), met een tijdsduur van een kwart seconde.

Syntaxis:
BEEP

Opmerkingen:
De instructies BEEP, CTRL-G en PRINT CHR$(7) hebben hetzelfde effect.

Voorbeeld:
```
2340 IF X>20 THEN BEEP
```

Zodra het getal X groter wordt dan 20, geeft de computer een pieptoon.

BLOAD opdracht

Functie:
Een geheugenbeeldbestand laden, beginnend op een opgegeven geheugenadres.

Syntaxis:
BLOAD *bestandsnaam*[,*startadres*]

Opmerkingen:
De *bestandsnaam* moet een geldige tekenreeks-expressie zijn, voorafgegaan door een aanduiding van de eenheid (bijvoorbeeld B: voor station B:) waar het bestand is opgeslagen.

Het *startadres* moet een geldige numerieke expressie zijn, met een waarde tussen 0 en 65535. Deze waarde geeft aan op welk adres binnen het segment uit de laatste DEF SEG-instructie het laden moet beginnen.

Als geen startadres wordt opgegeven, gaat het systeem uit van de waarde die in de laatste BSAVE-instructie werd gebruikt. In dat geval begint het laden op het adres waar het bestand werd opgeslagen.

Attentie:

Bij een BLOAD-opdracht wordt geen test uitgevoerd op de inhoud van de opgegeven adressen. U kunt bij BLOAD elk willekeurig geheugenadres opgeven. Er kunnen echter fouten optreden als u een adres opgeeft dat deel uitmaakt van de stack of de variabelenruimte voor GW-BASIC of een adres waar zich een GW-BASIC programma bevindt.

BLOAD en BSAVE zijn zeer nuttig voor het laden, resp. opslaan van machinetaalprogramma's, maar dat is niet hun enige functie. Via de opdracht DEF SEG kunt u bepalen op welk segment de commando's BLOAD en BSAVE betrekking hebben. Op die wijze kunt u bijvoorbeeld de schermbuffer vanaf diskette laden of op diskette opslaan. Bovendien dienen BLOAD en BSAVE voor het laden, resp. opslaan van grafische patronen.

Voorbeeld:

```
10 DEF SEG=&HB800
20 BLOAD"FIGUUR",0
```

(Dit voorbeeld werkt niet bij alle scherm-modi.)

DEF SEG op regel 10 verwijst naar het segment van de schermbuffer.

Samen met de DEF SEG-instructie op regel 10 zorgt het getal 0 voor het startadres, op regel 20, dat het laden op het juiste geheugenadres begint.

Het BLOAD-commando op regel 20 laadt het bestand *figuur* in de schermbuffer.

Attentie:

De nu volgende informatie over het commando BSAVE geeft aan hoe u het bestand *figuur* wordt opgeslagen.

BSAVE commando

Functie:

Gegevens uit bepaald deel van het geheugen opslaan in een opgegeven geheugenmedium.

Syntaxis:

BSAVE *bestandsnaam,startadres,lengte*

Opmerkingen:

De *bestandsnaam* moet een geldige tekenreeks-expressie zijn die de bestandsnaam bevat. Als de bestandsnaam niet minimaal één teken bevat, verschijnt de foutmelding "Onjuist bestandsnummer" en wordt de opdracht afgebroken.

Het *startadres* moet een geldige numerieke expressie zijn, met een waarde tussen 0 en 65535. Deze waarde geeft aan op welk adres binnen het segment uit de laatste DEF SEG-instructie het opslaan moet beginnen.

Ook *lengte* moet bestaan uit een geldige numerieke expressie met een waarde tussen 0 en 65535. De waarde bepaalt het aantal geheugenadressen waarvan de inhoud moet worden opgeslagen in een ander geheugenmedium.

Laat een BSAVE-commando altijd voorafgaan door een DEF SEG-instructie. Het adres uit de laatste DEF SEG-instructie wordt gebruikt om te bepalen waar het bestand wordt opgeslagen. Met de DEF SEG-opdracht wordt het segmentadres ingesteld op het begin van de schermbuffer. Een startadres van 0 en een lengte van 16384 zorgen ervoor dat de gehele schermbuffer van 16K wordt opgeslagen.

Voorbeeld:

```
10 DEF SEG=&HB800
20 BSAVE"FIGUUR",0,16384
```

DEF SEG op regel 10 verwijst naar het segment waar de schermbuffer zich bevindt.

Het BSAVE-commando op regel 20 slaat de schermbuffer op onder de bestandsnaam *figuur*.

CALL opdracht

Functie:

Aanroepen van een subroutine die in assembleer- of in machinetaal is geschreven.

Syntaxis:

CALL *numvar*[(*variabelen*)]

Opmerkingen:

Met *numvar* wordt het startadres van de aangeroepen subroutine aangegeven.

In *variabelen* worden de variabelen of constanten (onderling gescheiden door komma's) opgegeven die door de subroutine moeten worden gebruikt. De opgegeven variabelen moeten tussen haakjes staan.

De CALL-opdracht wordt veel gebruikt om vanuit GW-BASIC assembleerprogramma's aan te roepen. Dit is ook mogelijk met de USR-instructie, maar CALL is compatibel met meerdere programmeertalen, leidt tot een beter leesbare source code en kan meerdere argumenten tegelijk aan de subroutine doorgeven.

Bij een CALL-instructie worden de volgende bewerkingen uitgevoerd:

- Elk parameter-adres in de variabele wordt in de geheugen-stack geplaatst. Dat adres bestaat uit een startadres van 2 bytes in het data-segment van GW-BASIC.
- Het codesegment (CS) van het terugkeeradres en de startwaarde worden in de stack geplaatst.
- De programma-uitvoering wordt overgedragen aan de subroutine. Deze subroutine bevindt zich in het segment dat in de laatste DEF SEG-opdracht is aangeduid, beginnend op het startadres dat in de variabelenaam is opgegeven.
- De routine bestuurt nu de programma-uitvoering. U kunt verwijzen naar parameters door de stack pointer (SP) in te stellen op de base pointer (BP) en aan BP een positieve drempelwaarde toe te voegen.
- De aangeroepen routine kan de inhoud van registers beschadigen.

- Het aangeroepen programma moet "weten" hoeveel variabelen aan de subroutine moeten worden doorgegeven. Variabelen worden aangegeven door aan BP een positieve waarde toe te voegen, aangenomen dat de huidige waarde van de stack pointer door de routine is opgeslagen in BP (d.w.z. de instructie MOV BP,SP).
- Het aangeroepen programma moet het type kennen van de numerieke variabelen die aan de subroutine worden doorgegeven.
- De aangeroepen routine moet een RET *n* uitvoeren. Daarbij verwijst *n* naar het aantal parameters in de variabele, vermenigvuldigd met 2. Dit is noodzakelijk om de stack in te stellen op het startpunt van de aanroep-procedure.
- Waarden worden naar GW-BASIC teruggestuurd door in de argumenten-lijst de naam van de variabele op te geven, waarin het resultaat moet worden opgeslagen.
- Als het argument uit een tekenreeks bestaat, dan verwijst de startwaarde voor parameters naar drie bytes die *string descriptor* worden genoemd. Byte 0 van de string descriptor bevat de lengte van de tekenreeks (waarde tussen 0 en 255). De bytes 1 en 2 zijn resp. de laagste en hoogste acht bits van het startadres van de tekenreeks in de tekenreeks-ruimte.
- Als het argument bestaat uit tekst tussen aanhalingstekens in het programma, dan verwijst de string descriptor naar programmatekst. Let op dat u een programma niet op die wijze per abuis verandert of verminkt. Om dit te voorkomen, is het raadzaam aan tekst-tekenreeksen +"" toe te voegen, zoals in dit voorbeeld:

  ```
  20 A$="letterlijke tekst"+""
  ```

  Door die toevoeging wordt de tekst-tekenreeks gekopieerd naar de tekenreeks-ruimte. Daarna kan de tekenreeks worden veranderd zonder het programma te beschadigen.

Attentie:

Het is toegestaan in een routine de inhoud van tekenreeksen te laten wijzigen. Verander echter niet de lengte van een tekenreeks, want GW-BASIC is niet in staat tekenreeksen op correcte wijze te wissen als de lengte van die tekenreeksen door externe routines is veranderd.

Appendix D geeft meer informatie over de opdracht CALL en de functie USR.

Voorbeeld 1:

```
100 DEF SEG=&H2000
110 ROUT=0
120 CALL ROUT (A,B$,C)
.
.
.
```

Regel 100 stelt de waarde voor segment in op het hexadecimale adres 2000. Op regel 110 wordt ROUT op 0 gezet, zodat via de CALL-opdracht op regel 120 de subroutine wordt uitgevoerd die zich bevindt op geheugenadres 2000:0 (segment &H2000, adres 0).

Voorbeeld 2:

Onderstaand programma in 8086 assembleertaal demonstreert het gebruik van aan de routine doorgegeven parameters en het opslaan daarvan in variabele C.

```
PUSH BP
MOV BP,SP          ; slaat waarde van SP op in BP
MOV BX,8[BP]       ; slaat adres van B$ op in CL
MOV CL,[BX]        ; slaat lengte van B$ op in CL
MOV DX,1[BX]       ; slaat adres van tekst van B$ op in DX
.
.
.
MOV SI,10 [BP]     ; slaat adres van A op in SI
MOV DI,6[BP]       ; slaat naar C verwijzende pointer op in
DI
MOVSW              ; slaat variabele A op in C
RET 6              ; slaat stack weer op en keert terug naar
                   ; BASIC
```

MOVSW kopieert slechts 2 bytes. Dit is voldoende als de variabelen A en C gehele getallen zijn. Voor getallen met enkele precisie moeten 4 bytes worden gekopieerd, voor getallen met dubbele precisie 8 bytes.

Voorbeeld 3:

```
100 DEF SEG=&H2000
110 OPTEL=&H7FA
120 CALL OPTEL(A,B$,C)
.
.
.
```

Regel 100 verwijst naar het segment dat begint op het hexadecimale adres 2000. De waarde van de variabele OPTEL wordt aan het adres toegevoegd door het minst significante woord na de DEF SEG-waarde vier bits naar links te schuiven. (Die zogeheten shift-bewerking wordt door de processor zelf uitgevoerd; het is geen GW-BASIC instructie.) De waarde van de variabele OPTEL wordt op regel 110 ingesteld op de hexadecimale waarde &H7FA. Daarom roept CALL de routine aan die begint op het hexadecimale adres 2000:7FA (absoluut adres is hex 207FA).

CDBL functie

Functie:

Numerieke waarde omzetten in een getal met dubbele precisie.

Syntaxis:

CDBL(x)

Opmerkingen:

x moet een geldige numerieke expressie zijn.

Voorbeeld:

```
10 A=454.12345678
20 PRINT A;CDBL(A)
RUN
454.1235       454.1234436035156
Ok
```

Dit programma geeft de waarde van variabele *A* met dubbele precisie.

Attentie:

Zie ook de informatie bij de functies CINT en CSNG, waarmee getallen worden omgezet in gehele getallen, resp. getallen met enkele precisie.

CHAIN opdracht

Functie:

Programmabesturing overdragen aan het opgegeven programma en daaraan tevens een aantal variabelen doorgeven.

Syntaxis:

CHAIN [MERGE] *bestandsnaam*[,[*regel*][,[ALL][,DELETE *bereik*]]]

Opmerkingen:

Door toevoeging van MERGE worden programmaregels uit het aangeroepen programma in het huidige programma ingevoegd (een zogeheten *overlay*).

Attentie:

MERGE kan alleen worden uitgevoerd als het aangeroepen programma een ASCII-bestand is (eerder opgeslagen met de optie *a*). Zie ook de beschrijving van het commando MERGE.

Bij *bestandsnaam* moet de naam van het aangeroepen programma worden opgegeven. Als geen bestandstype wordt opgegeven, gaat het systeem er automatisch van uit dat het een programma met het bestandstype .BAS (voor BASIC) is.

Bij *regel* moet een regelnummer worden opgegeven dat in het aangeroepen programma voorkomt, of een expressie waarvan de uitkomst overeenkomt met een regelnummer in dat programma. Het aangeroepen programma wordt vanaf dat regelnummer uitgevoerd. In het volgende voorbeeld wordt het aangeroepen programma "PROGR1" uitgevoerd vanaf regel 1000:

```
CHAIN "PROGR1",1000
```

Als geen waarde voor *regel* wordt opgegeven, wordt het aangeroepen programma vanaf de eerste programmaregel uitgevoerd.

De waarde van *regel* wordt niet beïnvloed door een RENUM commando. De regelnummers in het opgegeven bereik worden wel door RENUM beïnvloed.

Met ALL geeft u aan dat alle variabelen uit het huidige programma aan het aangeroepen programma moeten worden doorgegeven. Bijvoorbeeld:

```
20 CHAIN "PROGR1",1000,ALL
```

Als u ALL achterwege laat, dan moet het huidige programma een COMMON-opdracht bevatten, die aangeeft welke variabelen aan het aangeroepen programma moeten worden doorgegeven.

CHAIN voert automatisch een RESTORE-instructie uit voordat het aangeroepen programma wordt uitgevoerd. Daarom begint een READ-opdracht met het inlezen van de eerste waarde in een DATA-opdracht. Het inlezen wordt dus niet hervat na het laatst ingelezen gegeven in een DATA-opdracht.

Zodra een *overlay* voor een speciaal doel is uitgevoerd, is het meestal raadzaam dat programma te wissen (met het commando DELETE), waarna indien gewenst een volgende overlay kan worden aangeroepen.

Een CHAIN MERGE-opdracht laat de bestanden open en laat ook de huidige instelling van OPTION BASE ongewijzigd. Als MERGE achterwege blijft, wordt de huidige OPTION BASE-instelling niet overgenomen. Bovendien geeft de CHAIN-opdracht zonder MERGE geen variabeletypen of zelf gedefinieerde functies door aan het aangeroepen programma. Dit heeft tot gevolg dat de gebruiker DEFINT-, DEFSNG-, DEFDBL-, DEFSTR- of DEF FN-opdrachten opnieuw moet invoeren in het aangeroepen programma als in die opdrachten variabelen worden gedefinieerd die door beide programma's gezamenlijk worden gebruikt.

Plaats zelf gedefinieerde functies in een programma altijd vóór een eventuele CHAIN MERGE-opdracht. Doet u dat omgekeerd, dan worden de definities ongedaan gemaakt door de uitvoering van de MERGE-instructie.

CHDIR commando

Functie:
Overgaan naar een andere directory.

Syntaxis:
CHDIR *padnaam*

Opmerkingen:
De *padnaam* is een tekenreeks-expressie van maximaal 63 tekens.

Stel dat station A: het standaardstation is. Als u dan *verkopen* tot de actieve directory voor station A: wilt maken en *voorraad* tot de actieve directory voor station B:, dan moet u het volgende intikken:

```
CHDIR "VERKOPEN"
CHDIR "B:VOORRAAD"
```

CHR$ functie

Functie:
Een ASCII-code omzetten in het bijbehorende teken.

Syntaxis:
CHR$(*n*)

Opmerkingen:

CHR$ wordt veel toegepast om een speciaal teken op het scherm te plaatsen of naar de printer te sturen. Zo kunt u bijvoorbeeld via de instructie PRINT CHR$(7) een pieptoon laten klinken, zoals vaak gebeurt om een foutmelding te accentueren. Via CHR$(12) wordt een form feed-instructie aan de printer doorgegeven.

Voorbeeld 1:

```
PRINT CHR$(66)
B
Ok
```

ASCII-code 66 komt overeen met hoofdletter B.

Voorbeeld 2:

```
PRINT CHR$(13)
```

Via deze instructie gaat de cursor naar de volgende regel.

CINT functie

Functie:

Getallen met decimalen afronden op het dichtstbijliggende gehele getal.

Syntaxis:

CINT(*x*)

Opmerkingen:

Als x kleiner is dan -32768 of groter dan 32767, dan volgt de foutmelding "Te groot getal."
Zie ook de functies FIX en INT voor inkorten tot gehele getallen.

Voorbeeld:

```
PRINT CINT(45.67)
46
Ok
```

Het getal 45.67 wordt naar boven afgerond tot het gehele getal 46.

CIRCLE opdracht

Functie:

Tekenen van cirkels, ellipsen en krommen als de grafische modus is ingeschakeld.

Syntaxis:

CIRCLE (*xmid*,*ymid*),*straal*[,[*kleur*][,[*begin*],[*eind*][,[*ratio*]]]

Opmerkingen:

De argumenten *xmid* en *ymid* bepalen de x- en y-coördinaat voor het middelpunt van een ellips. Het argument *straal* geeft de straal van de ellips (gemeten langs de langste as). De waarden voor *xmid* en *ymid* kunnen bestaan uit de uitkomst van een expressie en mogen verwijzen naar absolute of relatieve coördinaten.

Het argument *kleur* bepaalt de kleur voor de afgebeelde ellips. De waarde is afhankelijk van de ingeschakelde scherm-modus. Bij de opdrachten COLOR en SCREEN vindt u meer informatie over het gebruik van kleuren bij de diverse scherm-modi.

Bij hoge resolutie geeft 0 de kleur zwart en 1 wit. De standaardwaarde tijdens hoge resolutie is 1.

De waarden voor de optionele argumenten *begin* en *eind* duiden een hoek in radialen aan (tussen -2pi en 2pi). Deze waarden bepalen welk deel van de ellips verschijnt. Als voor *begin* of *eind* een negatieve waarde wordt opgegeven, wordt een verbindingslijn tussen de ellips en haar middelpunt getekend en worden de hoekwaarden verwerkt als bij een positieve waarde. (Dit is niet hetzelfde als vermeerderen met 2*pi!)

Het argument *ratio* bepaalt de verhouding tussen de x- en y-as van de ellips (x:y). De standaard *ratio* is afhankelijk van de scherm-modus, maar geeft een cirkel in elke grafische modus (uitgaande van een standaardmonitor met een schermratio, breedte:lengte, van 4:3).

Als de *ratio* kleiner dan 1 is, dan verschijnt een rij pixels, bij een ratio-waarde groter dan 1 verschijnt een kolom pixels.

In veel gevallen geeft een *ratio* van 1 tijdens medium resolutie betere ellipsen, die bovendien sneller worden getekend. De beginhoek mag kleiner zijn dan de eindhoek.

Voorbeeld 1:

```
10 SCREEN 1: CIRCLE(100,100),50
```

Dit programma tekent een cirkel met een straal van 50, waarvan het middelpunt ligt op de grafische coördinaten x=100 en y=100.

Voorbeeld 2:

```
1 'Programma voor 17 ellipsen
10 CLS
20 SCREEN 1
30 FOR R=160 TO 0 STEP-10
40 CIRCLE (160,100),R,,,,5/18
50 NEXT
```

Voorbeeld 3:

```
10 REM Programma tekent 5 bollen
20 GOTO 160
50 IF VERT GOTO 100
60 CIRCLE (X,Y),R,C,,,.07
70 FOR I = 1 TO 5
80 CIRCLE (X,Y),R,C,,,I*0.2:NEXT I
90 IF VERT THEN RETURN
100 CIRCLE (X,Y),R,C,,,1.3
110 CIRCLE (X,Y),R,C,,,1.9
120 CIRCLE (X,Y),R,C,,,3.6
130 CIRCLE (X,Y),R,C,,,9.8
140 IF VERT GOTO 60
150 RETURN
160 CLS:SCREEN 1:COLOR 0,1:KEY OFF:VERT=0
170 X=160:Y=100:C=1:R=50:GOSUB 50
180 X=30:Y=30:C=2:R=30:GOSUB 50
190 X=30:Y=169:GOSUB 50
200 X=289:Y=30:GOSUB 50
210 X=289:Y=169:GOSUB 50
220 LINE (30,30)-(289,169),1
230 LINE (30,169)-(289,30),1
240 LINE (30,169)-(289,30),1,B
250 Z$=INKEY$: IF Z$="" THEN 250
```

CLEAR commando

Functie:

Alle numerieke variabelen op 0 zetten, alle tekenreeksen leegmaken (nul-strings) en alle geopende bestanden sluiten. Optionele argumenten bepalen het einde van het geheugen en reserveren de tekenreeks- en stack-ruimte die door GW-BASIC kan worden gebruikt.

Syntaxis:

CLEAR [,[*expressie1*][,*expressie2*]]

Opmerkingen:

Het eerste argument, *expressie1*, verwijst naar een geheugenadres dat, indien het wordt opgegeven, het maximale aantal bytes bepaalt dat voor GW-BASIC beschikbaar is.

Het tweede argument, *expressie2*, reserveert stack-ruimte voor GW-BASIC. De standaardhoeveelheid is de laatst opgegeven stack-ruimte. Na het opstarten van GW-BASIC is de stack ingesteld op 512 bytes, ofwel op 1/8 van de totale geheugenruimte. De kleinste van deze twee waarden wordt ingesteld.

GW-BASIC hanteert een dynamische methode voor het toekennen van geheugenruimte. Alleen als alle voor GW-BASIC beschikbare geheugenruimte in gebruik is, treedt de fout "Geen ruimte meer voor tekenreeksen" op.

Het CLEAR-commando:

- Sluit alle geopende bestanden.
- Maakt alle COMMON-variabelen leeg.
- Stelt tekenreeks- en stack-ruimte in op standaardwaarde.
- Heft alle schijfbuffers op.
- Breekt geluidssignalen af.
- Activeert voorgrondmuziek.
- Voert PEN OFF *uit*.
- Voert STRIG OFF *uit*.
- Schakelt ON ERROR uit.

Voorbeeld 1:

```
CLEAR
```

Zet alle variabelen op 0 en maakt alle tekenreeksen leeg.

Voorbeeld 2:

```
CLEAR 32768
```

Zet alle variabelen op 0, maakt alle tekenreeksen leeg, beschermt geheugenadressen groter dan 32768. De stack-ruimte blijft ongewijzigd.

Voorbeeld 3:

```
CLEAR ,,2000
```

Zet alle variabelen op 0, maakt alle tekenreeksen leeg, reserveert 2000 bytes stack-ruimte en stelt alle beschikbare geheugenruimte in het segment ter beschikking.

Voorbeeld 4:

```
CLEAR ,32768,2000
```

Zet alle variabelen op 0, maakt alle tekenreeksen leeg, beschermt geheugenadressen groter dan 32768 en reserveert 2000 bytes stack-ruimte.

CLOSE opdracht

Functie:

Gegevenstransport met een bestand of randapparaat afsluiten.

Syntaxis:

CLOSE [[#]*bestandsnummer*[,[*bestandsnummer*]...]

Opmerkingen:

Het *bestandsnummer* moet het nummer zijn dat bij het openen van het bestand werd toegekend.

Na het uitvoeren van een CLOSE-opdracht is de samenhang tussen het toegekende bestandsnummer en het bestand zelf opgeheven. Het bestand of het randapparaat kan later worden heropend door toekenning van hetzelfde of een ander nummer.

Een CLOSE-opdracht zonder bestandsnummer sluit alle geopende bestanden en randapparaten.

Een CLOSE-opdracht die naar een bestand of systeemcomponent wordt gestuurd die geopend is voor sequentiële uitvoer, schrijft de laatste inhoud van de uitvoerbuffer naar dat bestand of dat randapparaat.

Ook de opdrachten END, NEW, RESET, SYSTEM of RUN en LOAD (zonder de toevoeging r) sluiten automatisch alle bestanden en randapparaten. Dit is niet het geval bij STOP.

Voorbeeld 1:

```
250 CLOSE
```

Sluit alle geopende bestanden en randapparaten.

Voorbeeld 2:

```
300 CLOSE 1,#2,#3
```

Sluit alle bestanden en randapparaten met de bestandsnummers 1, 2 en 3.

CLS opdracht

Functie:

Het beeldscherm leegmaken.

Syntaxis:

CLS [n]

Opmerkingen:

Voor *n* kunnen onderstaande waarden worden opgegeven:

n waarde	Effect
0	Verwijdert alle tekst en grafische patronen van het scherm.
1	Maakt alleen het grafisch venster leeg.
2	Maakt alleen het tekstvenster leeg.

Als een grafisch venster is geactiveerd, maakt een CLS-instructie zonder argumenten alleen het venster leeg. Wanneer geen grafisch venster is geopend, maakt CLS het tekstvenster leeg.

Als voor het scherm de alfanumerieke modus is geactiveerd, wordt de schermpagina opgevuld met de op dat moment gekozen achtergrondkleur (zie ook de opdrachten SCREEN en COLOR).

Als de grafische modus is ingeschakeld, wordt de gehele schermbuffer opgevuld met de achtergrondkleur.

Ook de toetsencombinatie CTRL-HOME maakt het scherm leeg. Hetzelfde gebeurt door het veranderen van de scherm-modus via SCREEN- of WIDTH-opdrachten.

CLS zet de cursor in de linkerbovenhoek van het scherm en maakt de laatst aangegeven coördinaat tot middelpunt van het scherm.

Na een VIEW-opdracht maakt CLS alleen het laatst opgegeven venster leeg.

Voorbeeld:

```
1 CLS
```

Maakt het beeldscherm leeg.

COLOR opdracht

Functie:

Kleur bepalen waarmee informatie op het scherm wordt weergegeven.

Syntaxis:

COLOR [*voorgrond*] [,[*achtergrond*] [,*kader*]]
COLOR [*achtergrond*] [,[*palet*]]
COLOR [*voorgrond*] [,[*achtergrond*]]

Opmerkingen:

In het algemeen bepaalt COLOR de voorgrond- en achtergrondkleur op het scherm. Via SCREEN 0 kan ook de kleur van een kader worden ingesteld. Bij SCREEN 1 kan geen voorgrondkleur worden gekozen voor grafische opdrachten. Onderstaand overzicht geeft de effecten van de verschillende syntaxis-versies tijdens diverse scherm-modi:

SCREEN 0
Wijzigt de huidige standaard voorgrond- en achtergrondkleuren voor tekstweergave. Bij *voorgrond* moet een geheel getal uit het bereik 0-31 worden opgegeven, eventueel als uitkomst van een expressie. Dit getal bepaalt de "voorgrondkleur" in de tekst-modus, die overeenkomt met de standaardkleur voor tekstweergave. Met de gehele getallen 0-15 kunnen 16 kleuren worden gekozen. Door bij het gekozen getal 16 op te tellen, wordt de informatie knipperend afgebeeld. Dus knipperend weergeven in kleur 7 wordt gerealiseerd door kleurnummer 23 op te geven.

Voor *achtergrond* kunt u een geheel getal tussen 0 en 7 opgeven waarmee de achtergrondkleur voor tekst wordt bepaald. De achtergrond kan niet knipperen.

Via een getal uit het bereik 0-15 geeft u de kleur voor een *kader* langs de randen van het scherm op. Ook voor een kader is knipperen onmogelijk.

Als u aan COLOR geen argumenten toevoegt, dan worden *achtergrond* en *kader* in de standaardkleur zwart (kleur 0) weergegeven. Zie de beschrijving van de opdracht SCREEN voor de beschrijving van de *voorgrondkleuren*.

SCREEN 1
In de scherm-modus 1 heeft COLOR een andere syntaxis, waaraan het argument *palet* is toegevoegd, bestaande uit een even of oneven geheel getal, eventueel als uitkomst van een expressie. Dit getal bepaalt uit welk palet de opgegeven kleuren worden gekozen.

Voor systemen zonder een Enhanced Graphics Adapter (EGA), of een met EGA compatibele video-adapter, komen de standaard kleurinstellingen voor het argument *palet* overeen met:

```
COLOR ,        'Gelijk aan de volgende drie PALETTE
               'opdrachten:
               '1 = groen, 2 = rood, 3 = geel

COLOR ,1       'Gelijk aan de volgende drie PALETTE
               'opdrachten:
               '1 = cyaan, 2 = magenta, 3 = hoge
               'intensiteit wit
```

Bij systemen met een EGA geldt echter:

```
COLOR ,0       'Gelijk aan de volgende drie PALETTE
               'opdrachten:
PALETTE 1,2    'kenmerk 1 = kleur 3 (groen)
PALETTE 2,4    'kenmerk 2 = kleur 5 (rood)
PALETTE 3,6    'kenmerk 3 = kleur 6 (bruin)

COLOR ,1       'Gelijk aan de volgende drie PALETTE
               'opdrachten:
PALETTE 1,3    'kenmerk 1 = kleur 3 (cyaan)
PALETTE 2,5    'kenmerk 2 = kleur 5 (magenta)
PALETTE 3,7    'kenmerk 3 = kleur 15 (wit)
```

Vergeet niet dat een COLOR-opdracht het effect van voorafgaande PALETTE-opdrachten ongedaan maakt.

SCREEN 2
Geen effect. Als u toch probeert COLOR tijdens deze scherm-modus te gebruiken, verschijnt de foutmelding "Onjuiste functieaanroep".

SCREEN 7-10
In deze scherm-modi kan geen kleur voor *kaders* worden opgegeven. De achtergrondkleur voor grafische patronen wordt bepaald door het argument *achtergrond*, dat een waarde moet hebben die voor de geactiveerde scherm-modus is toegestaan. Raadpleeg de informatie bij de SCREEN-opdracht voor meer bijzonderheden. Het argument *voorgrond* bepaalt de standaardkleur voor het tekenen van lijnen.

Als voor een argument een waarde buiten het toegestane bereik is opgegeven, volgt de foutmelding "Onjuiste functieaanroep".

Als de kleuren van voor- en achtergrond gelijk zijn, wordt weergegeven informatie onzichtbaar. De standaardkleur voor de achtergrond is zwart (kleur 0) voor alle systeemconfiguraties en alle scherm-modi.

Voor systemen met EGA biedt PALETTE een hoge mate van flexibiliteit bij het toewijzen van verschillende schermkleuren aan *kader*, *voorgrond* en *achtergrond*. Zie PALETTE voor meer bijzonderheden.

Zie voor meer informatie de beschrijvingen van CIRCLE, DRAW, LINE, PAINT, PALETTE, PRESET, PSET en SCREEN.

Nu volgen enige voorbeelden van COLOR-opdrachten en de effecten bij verschillende scherm-modi:

```
SCREEN 0
COLOR 1, 2, 3    'voorgrond=1, achtergrond=2, kader=3

SCREEN 1
COLOR 1,0        'voorgrond=1, palet=een even getal
COLOR 2,1        'voorgrond=2, palet=een oneven getal

SCREEN 7
COLOR 3,5        'voorgrond=3, achtergrond=5

SCREEN 8
COLOR 6,7        'voorgrond=6, achtergrond=7

SCREEN 9
COLOR 1,2        'voorgrond=1, achtergrond=2
```

COM(n) opdracht

Functie:
In- of uitschakelen van het detecteren van communicatie via een opgegeven communicatiepoort.

Syntaxis:
COM(*n*) ON
COM(*n*) OFF
COM(*n*) STOP

Opmerkingen:

Met *n* geeft u het nummer van communicatiepoort 1 of 2 op.

Gebruik een COM(*n*) ON-opdracht alvorens de detectie in te schakelen met ON COM(*n*). Als u na een COM(*n*) ON-opdracht in een ON COM(*n*)-opdracht een getal opgeeft dat niet gelijk is aan 0, dan controleert BASIC bij elke nieuwe opdracht of via de opgegeven communicatiepoort tekens worden aangeboden.

Na uitvoering van een COM(*n*) OFF-opdracht wordt niet langer gedetecteerd dat via de communicatiepoort gegevens binnenkomen en gaat die informatie dus verloren.

Ook na uitvoering van de opdracht COM(*n*) STOP reageert het systeem niet meer op communicatie. Als op de opgegeven poort informatie wordt aangeboden, onthoudt het systeem dit echter en reageert onmiddellijk na de eerstvolgende COM(*n*) ON-opdracht.

COMMON opdracht

Functie:

Doorgeven van variabelen aan een via CHAIN aangeroepen programma.

Syntaxis:

COMMON *variabelen*

Opmerkingen:

In het argument *variabelen* geeft u een of meer variabelenamen op, onderling gescheiden door komma's, die u wilt doorgeven aan het aangeroepen programma.

De COMMON-opdracht wordt gebruikt in combinatie met CHAIN.

COMMON-opdrachten kunnen op willekeurige posities in het programma staan, maar het is raadzaam ze aan het begin te plaatsen.

Een programma mag meerdere COMMON-opdrachten bevatten, maar elke variabele mag slechts in één van de COMMON-opdrachten worden genoemd. Om in een CHAIN-opdracht alle variabelen aan het aangeroepen programma door te geven, gebruikt u in die opdracht de optie ALL en laat u COMMON-opdrachten achterwege.

Als u matrix-variabelen wilt opgeven, duidt u dat aan door achter de variabelenaam haakjes te plaatsen.

Voorbeeld:

```
100 COMMON A, B, C, D(), G$
110 CHAIN "A:PROGRAM3"
```

Dit voorbeeld roept via CHAIN het programma "PROGRAM3" aan, dat op de diskette in station A: is opgeslagen. Aan dat programma worden de variabelen A, B en C, de matrix-variabele D en de tekenreeks-variabele G$ doorgegeven.

CONT commando

Functie:
Uitvoering van het programma voortzetten na een onderbreking.

Syntaxis:
CONT

Opmerkingen:
Nadat het programma is onderbroken door STOP, END of CTRL-BREAK, zorgt een CONT-commando ervoor dat de uitvoering van het programma wordt hervat vanaf het punt waar deze werd onderbroken. Als het programma werd onderbroken tijdens de uitvoering van een INPUT-opdracht, zal na een CONT-commando de prompt opnieuw worden afgebeeld.

CONT is een zeer nuttig hulpmiddel tijdens het testen van programma's. U kunt een deel van het programma laten uitvoeren tot een STOP-opdracht, daarna fouten verbeteren tijdens de commando-modus en vervolgens de uitvoering hervatten met CONT. U kunt de programma-uitvoering ook vanaf een andere regel laten hervatten door in de commando-modus een GOTO-opdracht op te geven.

Als een programmaregel wordt gewijzigd, werkt CONT niet meer.

COS functie

Functie:
Berekening van de cosinus voor een hoek in radialen.

Syntaxis:
COS(*x*)

Opmerkingen:
De waarde van x verwijst naar radialen. COS is de trigonometrische cosinusfunctie. U kunt graden omzetten in radialen door te vermenigvuldigen met pi/180.

COS(*x*) wordt berekend met enkele precisie, behalve wanneer eerder bij het opstarten van GW-BASIC /d is opgegeven om de dubbele precisie te activeren.

Voorbeeld 1:
```
10 X=2*COS(.4)
20 PRINT X
RUN
1.842122
Ok
```

Voorbeeld 2:
```
10 PI=3.141593
20 PRINT COS(PI)
30 GRADEN=180
40 RADIAL=GRADEN*PI/180
50 PRINT COS(RADIAL)
RUN
-1
-1
Ok
```

CSNG functie

Functie:
Een getal omzetten in een getal met enkele precisie.

Syntaxis:
CSNG(*x*)

Opmerkingen:
Voor *x* moet een numerieke expressie worden opgegeven (zie ook CINT en CDBL).

Voorbeeld:

```
10 A#=975.341222#
20 PRINT A#; CSNG(A#)
RUN
975.341222    975.3412
Ok
```

CSRLIN variabele

Functie:
Aangeven op welke schermregel de cursor zich bevindt.

Syntaxis:
y=CSRLIN

Opmerkingen:
Het regelnummer wordt opgeslagen in de variabele *y*. Het resultaat van CSRLIN is een getal uit het bereik 1-25.

CSRLIN geeft de verticale coördinaat (schermregel, rij) van de cursorpositie op de geactiveerde pagina. (Zie ook de SCREEN-opdracht.)

Met behulp van x=POS(0) kunt u nagaan op welke horizontale coördinaat (kolom) de cursor staat. Deze instructie leidt tot een waarde tussen 1 en 40, ofwel 1 en 80, afhankelijk van de op dat moment ingestelde schermbreedte (zie POS).

Voorbeeld:

```
10 Y = CSRLIN
20 X = POS(0)
30 LOCATE 24,1
40 PRINT "hallo"
50 LOCATE Y,X
RUN
hallo
Ok
```

CSRLIN op regel 10 slaat het huidige regelnummer op in variabele X en POS op regel 20 slaat het huidige kolomnummer op in Y. De instructie op regel 30 stuurt de cursor naar positie 1 op schermregel 24. Regel 40 beeldt op die positie het woord "hallo" af. De opdracht LOCATE op regel 50 verplaatst de cursor naar zijn oorspronkelijke positie.

CVI, CVS, CVD functies

Functie:
Tekenreeksen omzetten in numerieke waarden.

Syntaxis:
CVI (*tekenreeks van 2 bytes*)
CVS (*tekenreeks van 4 bytes*)
CVD (*tekenreeks van 8 bytes*)

Opmerkingen:
Numerieke waarden uit een onbepaald benaderbaar bestand worden ingelezen als tekenreeksen. Om ze in berekeningen te kunnen gebruiken, moeten die tekenreeksen dan ook weer worden omgezet in getallen.

CVI zet een tekenreeks van 2 bytes om in een geheel getal. De functie MKI$ voert de omgekeerde bewerking uit.

CVS zet een tekenreeks van 4 bytes om in een getal met enkele precisie. De functie MKS$ voert de omgekeerde bewerking uit.

CVD zet een tekenreeks van 8 bytes om in een getal met dubbele precisie. De functie MKD$ voert de omgekeerde bewerking uit.

(Zie MKI$, MKS$, MKD$)

Voorbeeld:

```
.
.
.
70 FIELD #1, 4 AS N$, 12 AS B$
80 GET #1
90 Y=CVS(N$)
.
.
.
```

Regel 80 leest een veld van regel #1 (dat veld is gedefinieerd in regel 70) en zet de eerste 4 bytes (die samen N$ vormen) om in een getal met enkele precisie, dat wordt opgeslagen in de variabele Y.

Een getal met enkele precisie kan soms wel 7 ASCII-codes (dus 7 bytes) lang zijn. Bij het wegschrijven van een bestand via een MKS$-conversie en het inlezen via een CVS-conversie, zijn voor elk verwerkt getal drie bytes noodzakelijk. Getallen met dubbele precisie, verwerkt via MKS$- en CVD-conversie, vereisen soms nog meer opslagruimte.

DATA opdracht

Functie:
Numerieke en tekenreeks-constanten opslaan die door een of meer READ-opdrachten worden gelezen.

Syntaxis:
DATA *constanten*

Opmerkingen:

Constanten zijn numerieke constante waarden, die van verschillend type mogen zijn, dus zowel gehele getallen als decimale getallen. De constanten in een DATA-opdracht worden onderling gescheiden door komma's. Expressies zijn niet toegestaan.

In een DATA-opdracht moeten alleen de constanten die zelf komma's, dubbele punten of significante spaties bevatten, worden omgeven door aanhalingstekens. In andere gevallen zijn aanhalingstekens overbodig.

DATA-opdrachten zijn op zich zelf geen uitvoerbare instructies. Zij hebben alleen effect in combinatie met READ-instructies. De DATA-opdrachten kunnen overal in het programma worden geplaatst. Een DATA-opdracht kan zoveel constanten bevatten als de regellengte toelaat. Er gelden geen beperkingen voor het aantal DATA-opdrachten in een programma.

Een READ-opdracht leest DATA-opdrachten in de volgorde waarin die in het programma staan, dus op basis van het regelnummer. De constanten in een DATA-opdracht kunnen worden beschouwd als één lange lijst gegevens, ongeacht hoe zij over meerdere DATA-regels zijn verdeeld en waar zij in het programma staan. Het variabele-type in de READ-opdracht moet overeenkomen met het type van de constante die vanuit de DATA-opdracht wordt ingelezen en in die variabele opgeslagen. Als dat niet het geval is, verschijnt de foutmelding "Onjuist type".

Met behulp van een RESTORE-opdracht kunnen dezelfde constanten in DATA-opdrachten meerdere malen worden gelezen.

Zie READ en RESTORE voor meer bijzonderheden en voorbeelden.

Voorbeeld 1:

```
.
.
.
80 FOR I= 1 TO 10
90 READ A(I)
100 NEXT I
110 DATA 3.08,5.19,3.12,3.98,4.22
120 DATA 5.08,5.55,4.00,3.16,3.37
.
.
.
```

Dit voorbeeldprogramma leest de waarden uit de DATA-opdrachten en slaat deze op in matrix A. Na uitvoering van de READ-opdracht is de waarde van A(1) gelijk aan 3.08, A(2) wordt 5.19, enz. De DATA-opdrachten (regels 110 en 120) mogen op elke gewenste plaats in het programma staan, zelfs voorafgaand aan de READ-opdracht.

Voorbeeld 2:

```
5 PRINT
10 PRINT "Naam", "Woonplaats", "Leeftijd"
20 READ N$,W$,L
30 DATA "Peters,","Rotterdam",45
40 PRINT N$, W$, L
RUN

Naam          Woonplaats        Leeftijd
Peters,       Rotterdam         45
Ok
```

Dit programma leest de tekenreeksen en numerieke gegevens uit de DATA-opdracht op regel 30 en geeft die op het scherm weer.

DATE$ opdracht en variabele

Functie:
Instellen of oproepen van de datum van heden.

Syntaxis als opdracht:
DATE$=v$

Syntaxis als variabele:
v$=DATE$

Opmerkingen:
In beide vormen van DATE$ moet v$ een geldige variabele of tekenreeks zijn. Voor de datum zijn de volgende formats toegestaan:

mm-dd-jj
mm/dd/jj
mm-dd-jjjj
mm/dd/jjjj

Als voor v$ een ongeldige tekenreeks wordt ingetikt, volgt de foutmelding "Onjuist type". De tot dan toe bestaande inhoud van v$ blijft in dat geval ongewijzigd.

Als een waarde buiten het toegestane bereik valt of ontbreekt, verschijnt de foutmelding "Onjuiste functieaanroep" en blijft de bestaande datum ongewijzigd.

De datum van heden, vastgelegd bij het initialiseren van het besturingssysteem, wordt opgehaald en in de tekenreeks-variabele v$ opgeslagen als DATE$ deel uitmaakt van een LET- of PRINT-opdracht.

Als DATE$ aan een variabele wordt toegekend, wordt de datum van heden in die variabele opgeslagen.

De instructie v$=DATE$ resulteert in een tekenreeks van 10 tekens, met het format mm-dd-jjjj. In mm staat de maand (bereik 01 t/m 12), in dd de dag (01 t/m 31) en in jjjj het jaartal (1980 t/m 2099).

Voorbeeld:
```
v$=DATE$
Ok
PRINT v$
01-25-1988
Ok
```

DEF FN opdracht

Functie:
Definiëren en benoemen van door de gebruiker opgestelde functies.

Syntaxis:
DEF FN*naam* [*argumenten*]*expressie*

Opmerkingen:

Voor *naam* moet een geldige variabelenaam worden opgegeven. Deze naam, voorafgegaan door FN, wordt de naam van de gehele functie.

Bij *argumenten* worden de variabelenamen opgegeven waarmee wordt gerekend als de functie wordt gebruikt. De afzonderlijke argumenten worden van elkaar gescheiden door komma's.

Voor *expressie* wordt een expressie ingetikt die de daadwerkelijke bewerking uitvoert. Er mag slechts één bewerking worden opgegeven.

In de DEF FN-opdracht dienen de argumenten slechts voor het definiëren van de functie. Zij hebben geen invloed op programmavariabelen met dezelfde naam. Een variabelenaam die in de definitie van een functie is gebruikt, mag ook in *argumenten* voorkomen. In dat geval wordt bij het aanroepen van de functie de waarde van de variabele gelezen. In andere gevallen wordt de huidige waarde van de variabele gebruikt.

De variabelen in *argumenten* worden bij het aanroepen van de functie, volgens een een-op-een basis, vervangen door argument-variabelen of -waarden.

Via DEF FN kunnen zowel tekenreeks-bewerkingen als numerieke bewerkingen worden gedefinieerd. Als de functienaam een type-aanduiding bevat, zal het resultaat van de expressie aan dat type worden aangepast voordat het wordt opgeslagen in de variabele waardoor de functie werd aangeroepen. Als in de functienaam een type-aanduiding is opgenomen, moet het argument van hetzelfde type zijn, anders treedt de fout "Onjuist type" op.

Een functie kan worden aangepast door hem opnieuw te definiëren in een volgend DEF FN-opdracht.

De DEF FN-opdracht moet worden uitgevoerd voordat de met die opdracht gedefinieerde functie kan worden aangeroepen. Als een functie wordt aangeroepen die niet vooraf in een DEF FN-opdracht werd gedefinieerd, volgt de foutmelding "Ongedefinieerde gebruikersfunctie".

DEF FN kan alleen in programma's worden toegepast en is dus niet toegestaan in de commando-modus.

Het is niet mogelijk functies binnen een DEF FN-opdracht herhaald te laten uitvoeren.

Voorbeeld:

```
.
.
400 R=1:S=2
410 DEF FNAB (X,Y)=X^3/Y^2
420 T=FNAB(R,S)
.
.
```

Regel 410 definieert de functie FNAB, die vervolgens op regel 420 wordt aangeroepen. Zodra de gedefinieerde functie is uitgevoerd, wordt het resultaat daarvan in variabele T opgeslagen. T krijgt dan de waarde van "R tot de macht 3, gedeeld door S tot de macht 2", ofwel 0,25.

DEFINT, DEFSNG, DEFDBL, DEFSTR opdrachten

Functie:
Bepaald type toekennen aan variabelen.

Syntaxis:
DEF*type letter(s)*

Opmerkingen:
Via deze DEF-opdrachten kan aan variabelen een van de volgende typen worden toegekend:

INT voor het type geheel getal (integer);
SNG voor een getal met enkele precisie (single precision);
DBL voor een getal met dubbele precisie (double precision);
STR voor een tekenreeks (0-255 tekens).

Bij letter(s) worden een of meer letters opgegeven, van elkaar gescheiden door komma's. Het is ook toegestaan een bepaald bereik van het alfabet op te geven.

De variabelen waarvan de naam begint met een van de in de DEF-opdracht opgegeven letters, nemen het type aan dat in DEF*type* is aangeduid. Als een van die variabelen echter elders in het programma is voorzien van een type-aanduiding (% ! # $) dan wordt de DEF-opdracht voor die variabele genegeerd.

Als een programma geen instructies bevat die een bepaald type aan variabelen toekennen, dan hanteert GW-BASIC voor alle variabelen het standaardtype enkele precisie.

Voorbeeld 1:
```
10 DEFDBL L-P
```

Alle variabelen waarvan de naam begint met L, M, N, O, of P worden variabelen met dubbele precisie.

Voorbeeld 2:
```
10 DEFSTR A
20 A="120#"
```

Alle variabelen waarvan de naam begint met de letter A, worden tekenreeks-variabelen. Het is daarna overbodig in het programma zelf aan de variabelenaam het teken $ toe te voegen.

Voorbeeld 3:
```
10 DEFINT I-N, W-Z
20 W$="120#"
```

Alle variabelen waarvan de naam begint met een van de letters I, J, K, L, M, N, W, X, Y, Z, worden gehele getallen (integers). De variabele W$ op regel 20 is een tekenreeks-variabele omdat aan de naam het teken $ is toegevoegd. Alle variabelen waarvan de naam met een W begint en zonder de toevoeging $, zullen echter als gehele getallen worden beschouwd.

DEF SEG opdracht

Functie:
Segmentadres instellen voor later volgende instructies BLOAD, BSAVE, CALL, PEEK, POKE of USR.

Syntaxis:
DEF SEG[*adres*]

Opmerkingen:
Voor *adres* moet een getal of een numerieke expressie met een uitkomst tussen 0 en 65535 worden opgegeven.

Het opgegeven adres wordt opgeslagen om te verwijzen naar een segment, wat noodzakelijk is in latere instructies als BLOAD, BSAVE, CALL, PEEK, POKE of USR.

Als het getal of de uitkomst van een expressie buiten het bereik 0-65535 ligt, dan volgt de foutmelding "Onjuiste functieaanroep". In dat geval wordt de bestaande waarde van *adres* gehandhaafd.

Als geen *adres* wordt opgegeven, dan wordt automatisch het data-segment (DS) van GW-BASIC gekozen. Dat adres fungeert dus als standaardwaarde.

Als u de optie *adres* opgeeft, moet deze binnen de grens van 16 bytes blijven.

Segmentadressen worden vier bits naar links verschoven; om het segmentadres te vinden, moet de geheugenlocatie dan ook door 16 worden gedeeld.

Voor BLOAD, BSAVE, CALL, PEEK en POKE voert de processor op de segmentadressen een shift-bewerking uit, waarbij de bits 4 posities naar links worden geschoven. Via deze bit-manipulatie wordt voor de eerstvolgende CALL-instructie het adres van het codesegment gegenereerd. Zie BLOAD, BSAVE, CALL, PEEK en POKE voor meer bijzonderheden.

GW-BASIC controleert niet of het gegenereerde segmentadres binnen het toegestane adressenbereik valt.

Voorbeeld 1:
```
10 DEF SEG=&HB800
```

Deze instructie kiest het segment van de schermbuffer.

Voorbeeld 2:
```
20 DEF SEG
```

Deze instructie kiest het data-segment (DS) van GW-BASIC.

Attentie:
Tussen DEF en SEG moet een spatie staan, anders beschouwt GW-BASIC "DEFSEG=100" als de opdracht om aan de variabele DEFSEG de waarde 100 toe te kennen.

DEF USR opdracht

Functie:

Startadres bepalen van een subroutine in assembleertaal, die via de USR - functie kan worden aangeroepen.

Syntaxis:

DEF USR[*n*]=*integer*

Opmerkingen:

Voor *n* kan een geheel getal uit het bereik 0-9 worden opgegeven. De waarde van *n* verwijst naar een adres. De eerstvolgende USR-instructie roept dan de routine aan die op dat adres begint. Als u geen waarde opgeeft, wordt automatisch DEF USR0 gekozen.

De waarde van het argument *integer* verwijst naar het startadres van de routine. Als in een programma meer dan 10 USR-routines nodig zijn, kan het aantal DEF USR[*n*]-opdrachten worden gebruikt dat noodzakelijk is om het startadres USR[*n*] voor elke routine opnieuw te definiëren.

Om het startadres van de routine te verkrijgen, telt u de waarde van het huidige segmentadres op bij integer.

Door het aanroepen van een routine in assembleertaal wordt de uitvoering van het GW-BASIC-programma onderbroken, omdat de programma-uitvoering wordt overgedragen aan de routine. Zodra de assembleerroutine is uitgevoerd, wordt teruggekeerd naar de positie in het BASIC programma waar de routine werd aangeroepen.

Voorbeeld:

```
.
.
.
190 DEF SEG=0
200 DEF USR0=24000
210 X=USR0(Y^2/2.89)
.
.
.
```

In de regels 190 en 200 wordt het absolute adres bepaald.

Regel 210 roept de USR-routine aan die zich op dat adres bevindt, en geeft de waarde van de expressie tussen haakjes, in de vorm van een geheel getal, door aan het gebruikersprogramma (zie ook USR).

Attentie:

Deze opdracht wordt vooral beschreven met het oog op compatibiliteit met andere GW-BASIC implementaties. De veelzijdige CALL-opdracht kan beter worden gebruikt als deze compatibiliteit niet van belang is.

DELETE commando

Functie:

Wissen van een of meer programmaregels.

Syntaxis:
DELETE [*regelnummer1*][-*regelnummer2*]
DELETE *regelnummer1*-

Opmerkingen:
Bij *regelnummer1* geeft u het nummer op van de eerste programmaregel die u wilt wissen.

Bij *regelnummer2* geeft u het nummer op van de laatste programmaregel die u wilt wissen.

Na DELETE moet minimaal één regelnummer volgen, anders verschijnt de foutmelding "Onjuiste functieaanroep".

Als u de regel wilt wissen die als laatste op het scherm is afgebeeld, mag u in plaats van het regelnummer een punt (.) intikken.

Na uitvoering van een DELETE-commando keert GW-BASIC altijd terug naar de commando-modus.

Voorbeeld 1:
```
DELETE 40
```

Wist de programmaregel met regelnummer 40.

Voorbeeld 2:
```
DELETE 40-100
```

Wist de regels 40 t/m 100.

Voorbeeld 3:
```
DELETE -40
```

Wist alle regels voorafgaand aan regel 40 en ook regel 40 zelf.

Voorbeeld 4:
```
DELETE 40-
```

Wist regel 40 en alle daaropvolgende programmaregels.

DIM opdracht

Functie:
Maximale afmetingen van matrix-dimensies bepalen en op basis daarvan de vereiste geheugenruimte reserveren.

Syntaxis:
DIM *variabele*(*dimensies*)[,*variabele*(*dimensies*)]...

Opmerkingen:
Als u een matrix-variabele gebruikt zonder deze vooraf met DIM te dimensioneren, dan hanteert GW-BASIC voor elke dimensie van die matrix 10 als maximale waarde. Zodra in dat geval voor een van de matrix-dimensies een grotere waarde wordt gebruikt, verschijnt de foutmelding "Te hoog indexgetal".

Voor elke dimensie van een matrix kan als maximale waarde 255 worden opgegeven. De minimale waarde voor elke dimensie bedraagt 0, behalve wanneer u met behulp van OPTION BASE de waarde 1 hebt ingesteld.

Een gedimensioneerde matrix kan binnen een programma niet voor een tweede keer worden gedimensioneerd. Dit is alleen mogelijk als vooraf een CLEAR- of ERASE-opdracht werd uitgevoerd.

Door uitvoering van een DIM-opdracht wordt in alle cellen van een matrix de waarde 0 opgeslagen.

Voorbeeld:

```
10 DIM A(20)
20 FOR I=0 TO 20
30 A(I)=I
40 NEXT I
```

In dit voorbeeld wordt eerst matrix A gedimensioneerd, waardoor tevens in elke cel van die matrix de waarde 0 wordt opgeslagen. Vervolgens wordt in elke cel van de matrix de waarde van I opgeslagen, dus 0 in A(0), 1 in A(1), enz. Als A een matrix-variabele met enkele precisie is (het standaardtype) dan zal de DIM-opdracht voor deze matrix 84 bytes geheugenruimte reserveren (21 cellen maal 4 bytes).

DRAW opdracht

Functie:

Tekenen van grafische afbeeldingen.

Syntaxis:

DRAW *tekenreeks-expressie*

Opmerkingen:

De DRAW-opdracht voegt veel van de mogelijkheden uit andere grafische instructies samen in een definitie-programma met de naam Graphics Macro Language (GML). GML-commando's bestaan uit één teken in een tekenreeks, eventueel gevolgd door een of meer argumenten.

De DRAW-opdracht kan alleen worden gebruikt tijdens de grafische modus.

Commando's voor grafische bewegingen

Elk van de volgende commando's begint een beweging vanaf de huidige grafische coördinaat. Meestal is dat de coördinaat van het laatste grafische punt dat werd geplot met LINE of PSET of met een ander GML-commando. Als een programma wordt uitgevoerd, fungeert het middelpunt van het beeldscherm als standaard huidige positie. De coördinaten van dat middelpunt zijn 160,100 bij medium resolutie en 320,100 bij hoge resolutie. De afstand van een beweging is gebaseerd op een schaalfactor vermenigvuldigd met een waarde *n*. De standaardwaarde voor *n* is 1. Als voor *n* geen andere waarde wordt opgegeven en bovendien de standaard schaalfactor wordt gebruikt, omvat de beweging dus één beeldpunt.

Commando	Bewegingsrichting
U*n*	omhoog
D*n*	omlaag
L*n*	naar links
R*n*	naar rechts
E*n*	diagonaal naar rechtsboven
F*n*	diagonaal naar rechtsonder
G*n*	diagonaal naar linksonder
H*n*	diagonaal naar linksboven

De bewegingen worden gestuurd door het argument:

Mx,y Absolute of relatieve beweging. Als x wordt voorafgegaan door + of -, dan worden x en y opgeteld bij de huidige grafische cursorpositie. Vervolgens tekent DRAW een lijn tussen de huidige en de nieuwe positie. Zonder voorafgaande + of - wordt een lijn getekend tussen de huidige positie en punt x,y.

Onderstaande commando's mogen vóór de bovengenoemde commando's worden geplaatst:

B Naar nieuwe positie springen, maar geen punten afbeelden.

N Naar nieuwe positie gaan en terugkeren naar uitgangspositie.

Ook de volgende commando's zijn toegestaan:

A*n* Stel een hoek in van *n* graden, waarbij *n* de volgende waarden kan hebben: 0 voor 0°, 1 voor 90°, 2 voor 180° of 3 voor 270°. Tekeningen die onder een hoek van 90° of 270° zijn geroteerd, worden zo geschaald dat hun afmeting bij een standaard schermratio van 4:3 even groot is als onder hoeken van 0° of 180°.

TA*n* Verander de bewegingsrichting onder een hoek van *n* graden, waarbij *n* een waarde tussen -360 en +360 kan hebben. Bij een positieve waarde verandert de richting tegen de wijzers van de klok in, bij een negatieve waarde met de klok mee.

C*n* Kies kleur *n*. Zie de opdrachten COLOR, PALETTE en SCREEN voor bijzonderheden over geldige kleuren, nummers en kenmerken.

S*n* Stel de schaalfactor in op *n*, waarbij *n* een waarde tussen 1 en 255 kan hebben, die na deling door 4 resulteert in de schaalfactor. De schaalfactor vermenigvuldigd met de afstanden in U, D, L, R, E, F, G, H of relatieve M commando's bepaalt de afstand die wordt afgelegd. De standaardwaarde voor S is 4.

x*tekenreeks*;*variabele* Met dit commando wordt een subtekenreeks uit een tekenreeks uitgevoerd, vergelijkbaar met een subroutine. Een tekenreeks kan een andere tekenreeks aanroepen, enz.

Het argument *tekenreeks* fungeert als variabele voor een aantal bewegingscommando's.

vulkleur;*kader* Bepaalt de kleuren voor grafische afbeeldingen en resulteert in ingekleurde vlakken.

Het argument *vulkleur* bepaalt met welke kleur vlakken worden ingekleurd.

Het argument *kader* bepaalt de kleur van de omtrek van vlakken.

Zie ook de opdrachten COLOR, PALETTE en SCREEN voor informatie over geldige kleuren, nummers en kenmerken.

Als u de argumenten *vulkleur* of *kader* gebruikt, moet u voor beide een waarde opgeven. Het commando *vulkleur;kader* vult vlakken niet op met patronen.

Numerieke argumenten

De syntaxis van deze argumenten kent twee vormen: "=*variabele*;", (*variabele* is een variabelenaam), of een constante (b.v. "123").

Bij de syntaxis "=*variabele*;" is de puntkomma verplicht!

U kunt variabelen ook specificeren met de functie VARPTR$(*variabele*).

Voorbeeld 1:

```
10 SCREEN 1
20 A=20
30 DRAW "U=A;R=A;D=A;L=A"
RUN
```

Dit voorbeeldprogramma tekent een kader in medium resolutie.

Voorbeeld 2:

Om een vierkant op het scherm te tekenen, moeten we rekening houden met de schermratio (de verhouding tussen breedte en hoogte van het scherm), die bij standaardbeeldschermen 4:3 is.

Om een vierkant te tekenen met zijden van 96 pixels op een scherm van 640 x 200 pixels (SCREEN 2) moeten de volgende berekeningen worden uitgevoerd:

Horizontaal = 96
Verticaal = 96*(200/640)*(4/3)

of:

Verticaal = 40
Horizontaal = 40*(640/200)*(3/4)

De horizontale waarde bedraagt 4/3 van de verticale waarde.

Voorbeeld 3:

```
10 CLS
20 SCREEN 1
30 PSET (60,125)
40 DRAW "E100; F100; L199"
RUN
```

Dit programma tekent een driehoek in medium resolutie.

EDIT commando

Functie:
Programmaregel op het scherm afbeelden en de cursor op het eerste cijfer van het regelnummer plaatsen, waarna de betreffende regel kan worden gewijzigd.

Syntaxis:
EDIT *regelnummer*
EDIT .

Opmerkingen:
Na EDIT moet een bestaand *regelnummer* worden opgegeven.

Als u het regelnummer vervangt door een punt (EDIT .), dan verschijnt de regel die als laatste op het scherm werd afgebeeld.

Als een programmaregel wordt ingetikt, wordt dat de huidige regel. Door

```
EDIT .
```

in te tikken, verschijnt die regel. De huidige regel is altijd de laatste regel waarnaar door een EDIT- of LIST-commando of door een foutmelding werd verwezen.

Als u voor het argument *regelnummer* een niet-bestaand regelnummer opgeeft, verschijnt de foutmelding "Ongedefinieerd regelnummer".

Voorbeeld:

```
EDIT 150
```

Regel 150 wordt afgebeeld en kan worden gewijzigd.

END opdracht

Functie:
Uitvoering van een programma beëindigen, alle bestanden sluiten en terugkeren naar de commando-modus.

Syntaxis:
END

Opmerkingen:
Een programma mag meerdere END-opdrachten bevatten.

Na uitvoering van END verschijnt geen melding "Onderbroken in regel *xxxx*" (*xxxx* is een regelnummer), zoals bij STOP wel het geval is.

Het is niet verplicht een programma af te sluiten met een END-opdracht. Na uitvoering van een END-opdracht keert GW-BASIC altijd terug naar de commando-modus.

END sluit alle geopende bestanden.

Voorbeeld:

```
520 IF K>1000 THEN END ELSE GOTO 20
```

Als de waarde van variabele K groter wordt dan 1000, wordt de programma-uitvoering afgebroken en gaat GW-BASIC over naar de commando-modus.

ENVIRON opdracht

Functie:

Mogelijkheid bieden tot wijzigen van gegevens in de environment tekenreeks tabel. Zo kan de pad-aanduiding voor een "child-process" worden gewijzigd (zie ENVIRON$, SHELL en het PATH commando in de MS-DOS utilities) of kunnen gegevens worden doorgegeven aan een "child" door het opstellen van een nieuwe environment tekenreeks.

Syntaxis:

ENVIRON *tekenreeks*

Opmerkingen:

Voor *tekenreeks* moet een geldige tekenreeks-expressie worden opgegeven die de nieuwe environment-tekenreeks-parameter bevat.

Het argument *tekenreeks* moet de volgende syntaxis hebben:

parnaam=tekst

waarbij *parnaam* bestaat uit de naam van de nieuwe parameter, bijvoorbeeld PATH.

Tussen *parnaam* en tekst moet een is gelijk teken (=) of een spatie staan. ENVIRON beschouwt alles links van "=" of de spatie als *parnaam* en alle tekens rechts ervan als tekst.

Het argument *tekst* verwijst naar het nieuwe gegeven. Als dat een nul-string of alleen een puntkomma is, dan wordt het gegeven (inclusief *parnaam*=) uit de environment-tekenreeks-tabel verwijderd en wordt die tabel gecomprimeerd. In *tekst* mogen geen lege tekenposities staan.

Als *parnaam* niet bestaat, dan wordt het argument *tekenreeks* aan het einde van de environment-tekenreeks-tabel toegevoegd.

Als *parnaam* wel bestaat, wordt deze gewist, de environment-tekenreeks-tabel wordt gecomprimeerd en de nieuwe tekenreeks wordt aan het eind van de tabel toegevoegd.

Voorbeelden:

Stel dat de environment-tekenreeks-tabel leeg is. Dan zal onderstaande instructie een standaardpad naar de hoofd-directory op station A: opstellen:

```
ENVIRON "PATH=A:\"
```

Als u werkt met een subdirectory *jan*, dan kunt u DEBUG uit de hoofd-directory halen.

U kunt een nieuwe parameter toevoegen:

```
ENVIRON "COMSPEC=A:\COMMAND.COM"
```

De nieuwe environment-tekenreeks-tabel bevat nu:

```
PATH=A:\;COMSPEC=A:\COMMAND.COM
```

Het pad kan nu worden gewijzigd:

```
ENVIRON "PATH=A:\VERKOPEN;A:\BOEKHOUD"
```

Met een combinatie van ENVIRON en de functie ENVIRON$ kan de pad-tekenreeks worden aangekoppeld:

```
ENVIRON "PATH="+ENVIRON$("PATH")+";B:\VOORBLD"
```

Ten slotte wissen we de parameter COMSPEC:

```
ENVIRON "COMSPEC=;"
```

In de environment-tekenreeks-tabel staat nu:

```
PATH=A:\VERKOPEN;A:\BOEKHOUD;B:\VOORBLD
```

ENVIRON$ functie

Functie:
Oproepen van een bepaalde environment-tekenreeks uit de environment-tekenreeks-tabel.

Syntaxis:
v$=ENVIRON$(*parnaam*)
v$=ENVIRON$(*parnum*)

Opmerkingen:
Het argument *parnaam* moet bestaan uit een geldige tekenreeks-expressie, waarmee u aangeeft welke parameter moet worden opgeroepen.

Voor *parnum* wordt een geheel getal (of uitkomst van een rekenkundige expressie) uit het bereik 1-255 opgegeven.

Bij een tekenreeks-argument haalt ENVIRON$ uit de environment-tekenreeks-tabel de tekst op die volgt op *parnaam*=.

Als *parnaam* niet bestaat, dan leidt ENVIRON$ tot een lege tekenreeks (nul-string).

Als de op te zoeken tekenreeks via een numeriek argument wordt aangeduid, haalt ENVIRON$ de tekenreeks op met het in *parnum* opgegeven rangnummer in de environment-tekenreeks-tabel.

Als dat rangnummer niet in de tabel voorkomt, resulteert de instructie in een nul-string.

De functie ENVIRON$ maakt onderscheid tussen hoofdletters en kleine letters.

Voorbeelden:

De instructies:

```
ENVIRON "PATH=A:\VERKOPEN;A:\BOEKHOUD;B:\MKT:" 'Ga
naar tabel en PRINT ENVIRON$("PATH") 'Geef
tekenreeks weer
```

laten op het scherm de volgende tekenreeks verschijnen:

```
A:\VERKOPEN;A:\BOEKHOUD;B:\MKT
```

De eerste tekenreeks uit de environment-tabel wordt weergegeven via de instructie:

```
PRINT ENVIRON$(1)
```

Het onderstaande programma slaat de environment-tekenreeks-tabel op in een matrix, zodat die tabel kan worden aangepast voor een "child-process". Na uitvoering daarvan wordt de environment-tekenreeks opnieuw opgeslagen.

```
10 DIM ENVTABEL$(10)
20 AANTPARS=1
30 WHILE LEN(ENVIRON$(AANTPARS)) #0
40 ENVTABEL$(AANTPARS) = ENVIRON$(AANTPARS)
50 AANTPARS=AANTPARS+1
60 WEND
70 AANTPARS=AANTPARS-1
72 WHILE LEN(ENVIRON$(1)) #0
73 A$=MID$(ENVIRON$(1),1,INSTR(ENVIRON$(1),"="))
74 ENVIRON A$=";"
75 WEND
90 ENVIRON "CHILDPAR1=SORT OP NAAM"
100 ENVIRON "CHILDPAR2=LIST OP NAAM"
.
.
.
1000 SHELL "CHILD" 'Voert "CHILD.EXE" uit
1002 WHILE LEN(ENVIRON$(1))#0
1003 A$=MID$(ENVIRON$(1),1,INSTR(ENVIRON$(1),"="))
1004 ENVIRON A$=";"
1005 WEND
1010 FOR I=1 TO AANTPARS
1020 ENVIRON ENVTABEL(I)
1030 NEXT I
.
.
.
```

De DIM opdracht op regel 10 wordt gebruikt omdat we veronderstellen dat er meer dan 10 gegevens zijn, zodat de matrix expliciet moet worden gedimensioneerd (zie beschrijving van DIM).

Op regel 20 stellen we het aantal gegevens in eerste instantie in op 1.

Op de regels 30 t/m 70 passen we de gegevensnummers aan.

De programmaregels 72 t/m 100 wissen de op dat moment gebruikte environment, maken een nieuwe versie aan en slaan die op. Regel 74 wist de tekenreeks.

Op de programmaregels 1000 t/m 1030 wordt een dergelijke procedure nogmaals uitgevoerd. Daardoor worden de eerder gewiste gegevens weer opgeslagen.

EOF functie

Functie:
Einde van een bestand detecteren. De waarde van EOF wordt -1 (waar, true) zodra het einde van een opgegeven (sequentieel of invoer/uitvoer) bestand is bereikt. Zolang dat niet het geval is, heeft EOF de waarde 0 (onwaar, false).

Syntaxis:
v=**EOF**(*bestandsnummer*)

Opmerkingen:
Als een GET-instructie wordt uitgevoerd, terwijl het einde van een bestand al is bereikt, geeft de functie EOF de waarde -1. Met behulp van een binaire zoekprocedure of een ander algoritme kan op die wijze de omvang van een bestand worden vastgesteld. Bij communicatie met geheugenmedia of randapparatuur, geeft EOF met de waarde -1 aan dat de communicatiebuffer leeg is.

EOF is zeer nuttig om het einde van een bestand te detecteren tijdens het invoeren van gegevens en zo te vermijden dat de fout "Bestandseinde gepasseerd" optreedt.

Voorbeeld:
```
10 OPEN "I",1,"DATA"
20 C=0
30 IF EOF (1) THEN 100
40 INPUT#1,M(C)
50 C=C+1;GOTO 30
100 END
```

Het bestand "DATA" wordt in matrix M ingelezen, tot het einde van dat bestand wordt bereikt. Zodra dat einde wordt gedetecteerd, krijgt EOF de waarde -1 (waar). In dat geval springt het programma naar regel 100.

ERASE opdracht

Functie:
Opheffen van eerder gedimensioneerde matrices.

Syntaxis:
ERASE *matrix-variabele(n)*

Opmerkingen:
Pas nadat een matrix met behulp van ERASE werd opgeheven, kan onder dezelfde variabelenaam een nieuw matrix worden gedimensioneerd. ERASE maakt tevens de door die matrix in beslag genomen geheugenruimte vrij.

Als u een matrix onder dezelfde naam opnieuw dimensioneert zonder deze vooraf met ERASE op te heffen, dan treedt een fout op.

Voorbeeld:

```
200 DIM B (250)
.
.
.
450 ERASE B,A
460 DIM B(3,4)
```

Op regel 450 worden de matrices B (gedimensioneerd op regel 200) en A opgeheven. Op regel 460 wordt opnieuw een nieuw matrix met de variabelenaam B gedimensioneerd. Dat nieuwe matrix telt 4 kolommen en 5 rijen, dus 20 cellen, waarin door DIM de waarde 0 wordt opgeslagen.

ERDEV, ERDEV$ variabelen

Functie:

ERDEV: foutcode aangeven van een fout die bij een onderdeel van het systeem is opgetreden. ERDEV$: aanduiden bij welk onderdeel van het systeem de fout is opgetreden.

Syntaxis:

ERDEV
ERDEV$

Opmerkingen:

ERDEV geeft de foutcode uit de laagste 8 bits van interrupt 24H. Bits 8 tot 15 van het "attribute-word" in het Device Header Block worden direct in de hoogste 8 bits geplaatst.

Als de fout betrekking heeft op het toetsenbord, de printer of het beeldscherm, dan bevat ERDEV$ een verwijzing met een lengte van 8 bytes. Heeft de fout betrekking op een ander onderdeel, dan slaat ERDEV$ de naam (A:, B:, e.d.) op in 2 bytes.

Voorbeeld:

Via INT 24H veroorzaakte printer-stuurprogramma LPT2: de fout "Geen papier".

ERDEV bevat dan in de laagste 8 bits de waarde 9 en de hoogste 8 bits bevatten het hoogste byte van de Device Header "word attributes".

```
ERDEV$ bevat de aanduiding "LPT2:    ".
```

Attentie:

Als het regelnummer niet rechts van de relationele operator staat, kan het niet door RENUM worden gewijzigd.

Omdat ERL en ERR gereserveerde woorden zijn, mogen geen van beiden links van een is-gelijk teken staan (dus niet worden gebruikt als variabelenaam, bijvoorbeeld bij een LET-opdracht).

ERROR opdracht

Functie:

Het optreden van een fout simuleren en foutcodes definiëren.

Syntaxis:

ERROR *geheel getal*

Opmerkingen:

ERROR moet worden gevolgd door een *geheel getal*, dat groter is dan 0 en kleiner dan 255.

Als daarvoor een getal wordt gebruikt dat in GW-BASIC al gereserveerd is voor een bepaalde fout, dan wordt die betreffende fout gesimuleerd en verschijnt de bijbehorende foutmelding.

De gebruiker moet voor zelf gedefinieerde fouten foutcodes kiezen die groter zijn dan de bestaande foutcodes in GW-BASIC. Op dit moment zijn 76 foutcodes gedefinieerd. Het is raadzaam een waarde te kiezen die ook nog geldig blijft bij uitbreiding van het aantal foutmeldingen in GW-BASIC.

U kunt zelf gedefinieerde fouten toepassen in routines voor foutafhandeling.

Als u in een ERROR-opdracht een foutcode gebruikt waarvoor geen foutmelding is gedefinieerd, volgt de melding "Niet afdrukbare fout".

Als u een ERROR-opdracht laat uitvoeren zonder de opgetreden fout daarna in een routine te laten afhandelen, dan verschijnt de bijbehorende foutmelding en wordt de uitvoering van het programma afgebroken.

Appendix A van deze handleiding bevat een overzicht van alle in GW-BASIC gedefinieerde foutcodes en bijbehorende foutmeldingen.

Voorbeeld 1:

```
LIST
10 S=10
20 T=5
30 ERROR S+T
40 END
Ok
RUN
Tekenreeks te lang in 30
Ok
```

Dit voorbeeld heeft betrekking op foutcode 15, die optreedt als een tekenreeks te lang is. Met dit voorbeeldprogramma zou u die fout tijdens de programma-modus kunnen simuleren.

Om dezelfde fout in de commando-modus te simuleren tikt u in:

```
ERROR 15
```

waarna de foutmelding verschijnt:

```
Tekenreeks te lang
Ok
```

Voorbeeld 2:

```
.
.
.
110 ON ERROR GOTO 400
120 INPUT "Hoeveel zet u in";B
130 IF B>5000 THEN ERROR 210
.
.
.
400 IF ERR=210 THEN PRINT "Inzet mag max. 5000
    gulden zijn"
410 IF ERL=130 THEN RESUME 120
.
.
.
```

Dit voorbeeld demonstreert een door de gebruiker gedefinieerde fout en de zelf geschreven foutafhandeling (regels 400-410).

EXP functie

Functie:

Machtsverheffing e^x uitvoeren (e is het grondtal van de natuurlijke logaritme).

Syntaxis:

EXP(x)

Opmerkingen:

De exponent x moet kleiner zijn dan 88.02969, omdat anders de foutmelding "Te groot getal" verschijnt. In dat geval wordt steeds de maximaal te berekenen waarde afgebeeld en gaat de uitvoering van het programma door.

EXP(x) wordt berekend met enkele precisie, behalve wanneer vooraf bij het opstarten van GW-BASIC /d is opgegeven om de dubbele precisie te activeren.

Voorbeeld:

```
10 X=5
20 PRINT EXP(X-1)
RUN
54.59815
Ok
```

Dit programma berekent de waarde van e tot de macht 4 (e^4).

EXTERR functie

Functie:

Uitgebreidere informatie over opgetreden fouten geven.

Syntaxis:

EXTERR(n)

Opmerkingen:

EXTERR geeft onder DOS 3.0 en hogere versies uitgebreidere informatie over fouten. Bij lagere versies van DOS geeft EXTERR altijd de foutcode 0. Het argument (*n*) moet 0, 1, 2 of 3 zijn, met resp. de volgende effecten:

n waarde	Resultaat
0	Uitgebreide foutcode
1	Uitgebreide foutclassificatie
2	Uitgebreide suggesties voor afhandeling
3	Uitgebreide foutenlocalisatie

De op het scherm verschijnende foutcodes zijn *niet* in GW-BASIC maar in DOS gedefinieerd. Raadpleeg de *MS-DOS Programmer's Reference* (versie 3.0 of later) voor een beschrijving van de DOS foutcodes.

De uitgebreide foutcode wordt in de praktijk steeds daadwerkelijk door GW-BASIC opgeroepen en weer opgeslagen als de betreffende DOS functies worden uitgevoerd. Bij het uitvoeren van een EXTERR-functie worden dus de opgeslagen waarden opgeroepen.

FIELD opdracht

Functie:

Ruimte voor variabelen reserveren in onbepaald benaderbare bestanden ("random access files").

Syntaxis:

FIELD [#]*bestandsnummer*, *lengte* AS *tekenreeksvar* [,*lengte* AS *tekenreeksvar*]...

Opmerkingen:

Het argument *bestandsnummer* is het bestandsnummer dat bij het openen van het bestand werd toegekend.

Het argument *lengte* bepaalt het aantal tekens dat voor de tekenreeksvariabele moet worden gereserveerd.

Bij *tekenreeksvar* wordt de variabelenaam opgegeven waarvan de inhoud uit het opgegeven bestand moet worden opgehaald of daarin moet worden opgeslagen.

Een FIELD-opdracht moet worden uitgevoerd om onderstaande operaties te kunnen uitvoeren:

- Gegevens uit een bestandsbuffer ophalen met een GET-opdracht;
- Gegevens invoeren die via PUT naar een bestand worden weggeschreven.

Onderstaande instructie reserveert in de bestandsbuffer de eerste 20 posities (bytes) voor de variabele NS, de 10 volgende posities voor IDS en de 40 daaropvolgende posities voor TELS:

```
FIELD 1, 20 AS N$, 10 AS ID$, 40 AS TEL$
```

Met FIELD wordt alleen ruimte gereserveerd; er vindt geen gegevenstransport van of naar de buffer plaats.

Het door FIELD in totaal gereserveerde aantal bytes mag uiteraard niet groter zijn dan de record-lengte die bij het openen van het aangeduide bestand werd vastgelegd. Bij overschrijding van die record-lengte treedt de fout "Field-lengte te groot" op. De standaard record-lengte is 128 bytes.

Voor elk bestand mogen meerdere FIELD-opdrachten worden gebruikt die tegelijkertijd actief zijn.

Attentie:

Gebruik in INPUT- of LET-instructies nooit variabelenamen die in een FIELD-opdracht zijn opgenomen. Zodra voor een variabele met behulp van een FIELD-opdracht ruimte is gereserveerd, verwijst de variabelenaam naar de juiste plaats in de bestandsbuffer. Als daarna een INPUT- of LET-instructie wordt uitgevoerd waarin die variabelenaam voorkomt, wordt de variabele-pointer naar de tekenreeks-ruimte verplaatst. Zie ook de beschrijving van de opdrachten LSET, RSET en GET.

FILES commando

Functie:

Een overzicht geven van de bestanden die op een schijf zijn opgeslagen.

Syntaxis:

FILES [*padnaam*]

Opmerkingen:

Als voor het argument *padnaam* niets wordt opgegeven, verschijnt een overzicht van alle bestanden in de huidige directory van de opgegeven schijf. In *padnaam* mogen vraagtekens (?) worden ingetikt als neutrale tekens voor tekens in de bestandsnaam of het bestandstype. Met een sterretje (*) als eerste teken in de bestandsnaam of het bestandstype geeft u aan dat elk bestand of bestandstype moet worden afgebeeld, ongeacht de naam of het bestandstype.

Het overzicht geeft tevens de naam van de directory en het aantal bytes in het bestand. Directories met een boomstructuur worden aangeduid met twee specifieke symbolen.

Subdirectories zijn gekenmerkt door <DIR> achter de directory-naam.

Voorbeelden:

```
FILES                (alle bestanden in huidige
                     directory)
FILES "*.BAS"        (alle bestanden met bestandstype
                     .BAS)
FILES "B:*.*"        (alle bestanden op schijf in
                     station B:)
FILES TEST?.BAS"     (alle bestanden met het
                     bestandstype .BAS en een naam
                     van vijf letters, waarvan de
                     eerste vier letters bestaan uit
                     TEST)
```

Aan FILES mogen padnamen worden toegevoegd, waarna de directory van dat pad wordt afgebeeld. Als geen padnaam wordt ingetikt, geeft het systeem de inhoud van de huidige directory.

```
FILES "BOEKHOUD\"
```

Met deze instructie verkrijgt u een overzicht van alle bestanden in de directory *boekhoud*.

```
FILES "B:BOEKHOUD\*.BET"
```

Met deze instructie verkrijgt u een overzicht van alle bestanden in de directory *boekhoud* die op de diskette in station B: zijn opgeslagen en die het bestandstype .BET (betaald) hebben.

FIX functie

Functie:
Bij decimale getallen de cijfers achter de decimale punt verwijderen.

Syntaxis:
FIX(*x*)

Opmerkingen:
FIX rondt decimale getallen niet af, maar kort deze in tot een geheel getal door de cijfers achter de decimale punt te verwijderen.

FIX(*x*) komt overeen met de bewerking SGN(*x*)*INT(ABS(*x*)). Het essentiële verschil tussen FIX en INT is dat FIX bij een negatieve waarde van *x* niet resulteert in afronden naar beneden.

FIX is nuttig bij het uitvoeren van modulo-bewerkingen.

Voorbeeld 1:

```
A=58.75:PRINT FIX(A)
58
Ok
```

Voorbeeld 2:

```
PRINT FIX(-58.75)
-58
Ok
```

FOR...NEXT opdrachten

Functie:
Herhaald uitvoeren van een reeks instructies, door middel van een programmalus, ook wel *loop* genoemd.

Syntaxis:
FOR *variabele=x* TO *y* [STEP *z*]
.
.
.
NEXT [*variabele*][,*variabele*]

Opmerkingen:
Het argument *variabele* dient als teller.

De argumenten *x*, *y* en *z* zijn numerieke expressies.

STEP *z* duidt aan met welke waarde de tellerstand moet worden gewijzigd elke keer nadat de lus is doorlopen.

De eerste waarde, *x*, geeft de beginstand van de lusteller op het moment waarop de programmalus voor de eerste keer zal worden doorlopen. De waarde van *y* bepaalt bij welke tellerstand de lus niet langer mag worden herhaald.

De instructies in de programmaregels tussen FOR en NEXT worden uitgevoerd. De uitvoering van NEXT hoogt de tellerstand op met de waarde die bij STEP voor *z* is opgegeven. Als STEP *z* niet wordt gebruikt, wordt de tellerstand steeds met 1 opgehoogd.

Na uitvoering van de NEXT-opdracht controleert het systeem of de nieuwe tellerstand groter is dan de opgegeven eindstand *y*. Is dat niet het geval, dan springt GW-BASIC terug naar de instructie die direct op de FOR-opdracht volgt. Vervolgens worden alle instructies tot NEXT opnieuw uitgevoerd. Zodra de tellerstand groter wordt dan *y*, gaat de uitvoering van het programma verder met de eerstvolgende instructie na de NEXT-opdracht.

De lusbewerking wordt niet uitgevoerd als de beginstand *x* van de teller (vermenigvuldigd met STEP *z*) groter is dan de eindstand *y* (vermenigvuldigd met STEP *z*).

Het is wel toegestaan de tellerstand steeds te verlagen door bij STEP *z* een negatieve waarde op te geven. In dat geval mag de beginstand van de teller groter zijn dan de eindstand. De tellerstand wordt steeds verlaagd met de voor *z* opgegeven waarde. De lusbewerking wordt herhaald tot de tellerstand kleiner dan *y* is geworden.

Nesten van lussen:

BASIC kent de mogelijkheid programmalussen te "nesten". Dit houdt in dat binnen een FOR...NEXT-lus een andere FOR...NEXT-lus wordt opgenomen. Daarbij moet elke lus een unieke variabelenaam voor de teller hebben en de NEXT-opdracht van de binnenste lus moet voorafgaan aan de NEXT-opdracht van de buitenste lus. Als twee of meer geneste lussen één gezamenlijk "eindpunt" hebben is één NEXT voor die lussen voldoende.

Achter NEXT mag de variabelenaam van de teller worden toegevoegd, maar dat is niet verplicht. Zonder variabelenaam zal het programma terugspringen naar de laatst uitgevoerde FOR-opdracht.

Als een NEXT-opdracht wordt uitgevoerd zonder dat vooraf de bijbehorende FOR-instructie is uitgevoerd, dan verschijnt de foutmelding "NEXT zonder FOR" en stopt de uitvoering van het programma.

Voorbeeld 1:

```
10 K=10
20 FOR I% = 1 TO K STEP 2
30 PRINT I%
60 NEXT
RUN
1
3
5
7
9
Ok
```

Dit voorbeeld geeft de waarden van de integer-variabele I%, vanaf de beginstand van de teller (1) en vervolgens steeds opgehoogd met de STEP-waarde (2). De teller is een integer-variabele, aangeduid door het percentteken (%), omdat gehele getallen sneller worden verwerkt dan decimale getallen.

Voorbeeld 2:

```
10 R=0
20 FOR S=1 TO R
30 PRINT S
40 NEXT S
```

De programmalus in dit voorbeeld wordt niet uitgevoerd omdat de beginstand S (1) al onmiddellijk groter is dan de eindstand R (0). Het programma zal geen enkele waarde van S op het scherm weergeven.

Voorbeeld 3:

```
10 S=5: FOR S=1 TO S+5: PRINT S;: NEXT
RUN
1    2    3    4    5    6    7    8    9    10
Ok
```

Deze lus wordt 10 keer doorlopen. De eindwaarde van de teller wordt steeds bepaald vóór de beginwaarde.

FRE functie

Functie:

Aangeven (in aantal bytes) hoeveel ruimte in het voor tekenreeksen gereserveerde geheugen nog vrij is.

Syntaxis:

FRE(*x$*)
FRE(*x*)

Opmerkingen:

De argumenten (*x$*) en (*x*) zijn zogeheten "dummy-argumenten", wat inhoudt dat het in principe onbelangrijk is welke waarde ervoor wordt ingetikt.

Alvorens GW-BASIC via de functie FRE(*x$*) aangeeft hoeveel ruimte in het geheugen nog vrij is voor tekenreeksen, wordt eerst "de bezem door dat geheugen gehaald". De in het geheugen opgeslagen gegevens worden zo effectief mogelijk gereorganiseerd. Tekenreeksen die in delen verspreid zijn opgeslagen, worden verplaatst, zodat ruimte vrij komt voor nieuwe gegevens.

Als FRE niet wordt gebruikt, zal GW-BASIC automatisch een soortgelijke "opruiming" uitvoeren zodra alle tekenreeks-ruimte in gebruik is. Deze procedure kan 1 tot anderhalve minuut in beslag nemen.

Wanneer bij FRE de tekenreeks ("") of een andere tekenreeks wordt opgegeven, geeft u expliciet opdracht tot het reorganiseren van de opgeslagen gegevens. Door regelmatig uitvoeren van FRE("") beperkt u de tijd die elke reorganisatie op zich vereist.

Tijdens het uitvoeren van de reorganisatie is het niet mogelijk de functie CTRL-BREAK te gebruiken.

Voorbeeld:

```
PRINT FRE(0)
14542
Ok
```

De bovenstaande waarde is slechts een voorbeeld.

GET opdracht (Bestanden I/O)

Functie:

Gegevens ophalen uit een onbepaald benaderbaar bestand en deze in een bestandsbuffer plaatsen.

Syntaxis:

GET [#]*bestandsnummer* [,*record-nummer*]

Opmerkingen:

Bij *bestandsnummer* geeft u het nummer op dat bij het openen van het bestand werd toegekend.

Het *record-nummer* verwijst naar een bepaald record. De waarde moet groter dan 0 en kleiner dan 16.777.215 zijn. Als u geen *record-nummer* opgeeft, leest GET het record in dat volgt op het record dat met de laatste GET-instructie is ingelezen.

Na een GET-opdracht kunnen via INPUT# en LINE INPUT# gegevens uit de bestandsbuffer worden ingelezen.

GET kan ook worden toegepast voor communicatie-bestanden, waarbij het argument [record-nummer] verwijst naar het aantal bytes dat uit de communicatiebuffer moet worden gelezen. Die waarde mag niet groter zijn dan de bufferlengte die is ingesteld in de opdracht OPEN COM(n).

Voorbeeld:

```
10 OPEN "R",1,"A:VERKOPER.FIL"
20 FIELD 1, 30 AS NAAM$, 120 AS ADRES$, 15 AS
   WOONPL$
30 GET1
40 PRINT NAAM$, ADRES$, WOONPL$
50 CLOSE 1
```

Dit voorbeeldprogramma opent het onbepaald benaderbaar bestand (random access file) *verkoper.fil*. Op regel 20 worden de velden gedefinieerd. GET op regel 30 leest een record uit het geopende bestand en zet dat in de bestandsbuffer. Regel 40 geeft de zojuist ingelezen informatie op het scherm weer en regel 50 sluit het bestand.

GET opdracht (Grafisch)

Functie:

Transporteren van grafische gegevens van het beeldscherm.

Syntaxis:

GET(x1,y1)-(x2,y2),*matrix-naam*

Opmerkingen:

De opdrachten PUT en GET dienen ook voor het versturen van grafische gegevens van en naar het beeldscherm. Deze opdrachten bieden tijdens de grafische modi mogelijkheden voor animaties en snel bewegende grafische afbeeldingen.

GET slaat in de opgegeven matrix de grafische scherminhoud op die zich bevindt binnen een vooraf gedefinieerd kader. De rechthoek binnen het kader is op dezelfde wijze gedefinieerd als rechthoeken die met de opdracht LINE en de toevoeging ,B worden getekend.

De matrix dient als opslagplaats voor de grafische gegevens. Voor dit doel mag geen tekenreeks-matrix worden gebruikt. De afmeting van de matrix moet groot genoeg zijn voor de gehele afbeelding. De met behulp van GET in de matrix opgeslagen gegevens zijn alleen leesbaar voor grafische modi (behalve wanneer een integer-matrix wordt gebruikt, zoals nog zal blijken).

De gegevens worden als volgt opgeslagen:

- 2 bytes voor het aantal bits van de x-dimensie
- 2 bytes voor het aantal bits van de y-dimensie
- de matrix-gegevens zelf

De gegevens voor elke rij pixels (beeldpunten) worden per byte links aangesloten. Als een aantal bits moet worden opgeslagen dat kleiner dan een veelvoud van 8 is, worden de resterende posities in de byte opgevuld met nullen. De vereiste afmeting van de matrix bedraagt in bytes:

4 + INT((x**bits per pixel*+7)/8) * y

Raadpleeg de beschrijving van de opdracht SCREEN voor de waarden van "bits per pixel" bij verschillende scherm-modi.

Het aantal bytes voor elke cel van de matrix bedraagt:

- 2 bytes per cel van een integer-matrix
- 4 bytes per cel van een matrix met enkele precisie
- 8 bytes per cel van een matrix met dubbele precisie

Een integer-matrix voor de opslag van een afbeelding van 10 bij 12 vereist 40 bytes, wat kan worden afgeleid uit de berekening 4+INT(10*2+7)/8)*12. Daarvoor is dus een matrix noodzakelijk van minimaal 20 cellen.

Als bij OPTION BASE de waarde 0 is opgegeven, kan een integer-matrix worden opgesteld voor het controleren van de dimensies x en y en de gegevens. De x-dimensie bevindt zich in cel 0 van de matrix en de y-dimensie in cel 1. Bij gehele getallen wordt eerst de laagste byte en daarna de hoogste byte opgeslagen, maar bij het transporteren van de gegevens is die volgorde omgekeerd; eerst de hoogste (meest linkse) byte, gevolgd door de laagste.

Het is mogelijk een afbeelding van de ene naar een andere modus over te brengen, maar dat kan wel tot vreemde resultaten leiden, omdat de beeldpunten per modus verschillen.

Voorbeeld:

```
10 CLS: SCREEN 1
20 PSET(130,120)
30 DRAW "U25;E7;R20;D32;L6;U12;L14"
40 DRAW "D12;L6": PSET(137,102)
50 DRAW "U4;E4;R8;D8;L12"
60 PSET(137,88)
70 DRAW "E4;R20;D32;G4":PAINT(139,87)
80 DIM A(500)
90 GET (125,130)-(170,80),A
100 FOR I=1 TO 1000:NEXT I
110 PUT (20,20),A,PSET
120 FOR I=1 TO 1000:NEXT I
130 GET (125,130)-(170,80),A
140 FOR I=1 TO 1000:NEXT I
150 PUT (220,130),A,PRESET
```

GOSUB...RETURN opdrachten

Functie:

Vanuit het programma naar een subroutine springen en weer terug.

Syntaxis:

GOSUB *<regelnummer1>*

.
.
.

RETURN [*<regelnummer2>*]

Opmerkingen:

Het argument *regelnummer1* achter GOSUB moet verwijzen naar de eerste regel van de subroutine.

Normaal gesproken wordt de uitvoering van het programma na een RETURN hervat bij de eerste opdracht na GOSUB. Door aan RETURN het argument *regelnummer2* toe te voegen, kan het programma echter naar een andere programmaregel worden gestuurd. Gebruik deze extra mogelijkheid echter zeer zorgvuldig, want het systeem onthoudt eventueel eerder uitgevoerde instructies als GOSUB, WHILE of FOR. Wanneer als gevolg van een sprong naar een bepaald regelnummer de bijbehorende RETURN-, WEND- of NEXT-instructies worden overgeslagen, treden nogal eens fouten op als "FOR zonder NEXT". In dat geval sprong u vanuit de subroutine terug naar een punt voorbij de NEXT-opdracht dat behoort bij een FOR-opdracht die kort voor GOSUB werd uitgevoerd. In routines voor het detecteren van gebeurtenissen mag u alleen een regelnummer achter RETURN plaatsen als u zeker weet dat slechts één bepaalde gebeurtenis kan optreden. Tijdens routines optredende gebeurtenissen laten namelijk op de stack onduidelijke informatie achter, wat kan leiden tot onvoorspelbaar gedrag van het programma.

U kunt een subroutine meerdere malen aanroepen. Het is ook toegestaan vanuit een subroutine een tweede subroutine aan te roepen. Dit wordt "nesten" van subroutines genoemd. De mogelijkheden hiertoe worden alleen beperkt door de omvang van de stack, die met CLEAR kan worden aangepast om het aantal geneste GOSUB-opdrachten te vergroten.

In één subroutine mogen meerdere RETURN-opdrachten staan. Bij een RETURN zonder regelnummer springt het programma terug naar de instructie die direct volgt op de laatst uitgevoerde GOSUB.

De plaats van subroutines in een programma is niet aan voorwaarden gebonden, maar het is raadzaam ze duidelijk van het hoofdprogramma te onderscheiden. Ook is het verstandig vlak vóór de subroutine STOP, END of GOTO op te nemen, zodat de subroutine niet per abuis wordt uitgevoerd.

De compiler beschikt over de mogelijkheid opdrachten van regel-labels te voorzien, zodat de volgende syntaxis toegestaan is:

GOSUB {*<regelnummer1>*|*<regel-label1>*}
.
.
.
RETURN [{*<regelnummer2>*|*<regel-label2>*}]

Attentie:
De voorgaande beschrijving van subroutines heeft alleen betrekking op GOSUB-opdrachten, niet op subroutines met de structuur SUB...END...SUB.

Voorbeeld:
Het onderstaande voorbeeld berekent de arcsinus voor waarden tussen -1 en 1. De arcsinus is de inverse van de sinus. De uitkomsten duiden op hoeken tussen pi/2 radialen en 3pi/2 radialen (90° en 270°). Compileer het programma via /N, zodat de regelnummers verdwijnen.

```
input "sinus";x
while (x < -1 or x > 1)
print "Sinus kan min. -1 en max. +1 zijn!"
input "sinus";x
wend
pi = 3.141593
gok2=pi/2: gok1=3*(pi/2)
while abs(gok1-gok2) > .0000005
gosub nieuw
wend
print "arcsin van ";x;" = "; tijdel
end

nieuw:
tijdel = (gok1 + gok2)/2
if sin(tijdel) > x then gok2= tijdel else
gok1=temp
return
```

Voorbeeld van weergeven uitkomst:

```
sinus? 3
Sinus kan min. -1 en max. +1 zijn!
sinus? .43
arcsin van .43 = 2.6971
```

Dit resultaat betekent dat een sinus van .43 hoort bij een hoek van 2.6971 radialen.

GOTO opdracht

Functie:
Onvoorwaardelijk naar de programmaregel met het opgegeven regelnummer springen.

Syntaxis:
GOTO *regelnummer*

Opmerkingen:
Voor het argument *regelnummer* geeft u een regelnummer op dat in het programma voorkomt.

Als op de regel met het opgegeven nummer een uitvoerbare opdracht staat, wordt de programma-uitvoering bij die opdracht hervat. In andere gevallen gaat de uitvoering van het programma verder bij de eerstvolgende uitvoerbare instructie.

Voorbeeld:

```
10 READ R
20 PRINT "Straal =";R,
30 OPP = 3.14*R^2
40 PRINT "Opp. =";OPP
50 GOTO 10
60 DATA 5,7,12
RUN
Straal = 5    Opp. = 78.5
Straal = 7    Opp. = 153.86
Straal = 12   Opp. = 452.16
Geen DATA meer 10
Ok
```

De foutmelding "Geen DATA meer 10" verschijnt omdat het programma in regel 50 wordt teruggestuurd naar regel 10 en daar de opdracht krijgt voor de vierde keer een DATA-waarde te lezen. Er zijn echter maar drie DATA-waarden.

HEX$ functie

Functie:
Decimale getallen omzetten in hexadecimale getallen.

Syntaxis:
v$=HEX$(x)

Opmerkingen:
HEX$ zet decimale getallen tussen -32768 en +65535 om in hexadecimale tekenreeksen uit het bereik 0 t/m FFFF.

Ons gangbare decimale stelsel is gebaseerd op het grondtal 10. Bij het hexadecimale stelsel wordt het grondtal 16 gebruikt. De appendices C en D geven meer bijzonderheden over hexadecimale getallen. Zie ook de beschrijving van OCT$ voor omzetting naar octale getallen (grondtal 8).

Het argument (x) wordt eerst afgerond tot een geheel getal en daarna omgezet in een hexadecimaal getal. Als (x) negatief is, wordt de zogeheten "2's complement" (binair) gebruikt.

Voorbeeld:

```
10 CLS: INPUT"Decimaal getal";X
20 A$=HEX$(X)
30 PRINT X "decimaal is "A$" hexadecimaal"
RUN
Decimaal getal? 32
32 decimaal is 20 hexadecimaal
Ok
```

Uitleg: hexadecimaal 20 betekent $2*16^1 + 0*16^0 = 2*16 + 0*1$

IF opdracht

Functie:

Voorwaarde vaststellen voor het nemen van een beslissing over het vervolg van de programma-uitvoering.

Syntaxis:

IF *expressie*[,] THEN *opdracht(en)* [,] [ELSE *opdracht(en)*]
IF *expressie*[,] GOTO *regelnummer* [[,] ELSE *opdracht(en)*]

Opmerkingen:

Als de uitkomst van het argument *expressie* ongelijk is aan "logisch 0" betekent dit dat aan de voorwaarde is voldaan, dat deze waar (true) is . Alleen in dit geval worden THEN of GOTO en de daaropvolgende instructies uitgevoerd. Als die uitkomst gelijk is aan "logisch 0", is hij onwaar (false). In dat geval worden THEN of GOTO en de daaropvolgende instructies genegeerd.

Als een ELSE-instructie is opgenomen wordt de direct daarop volgende instructie uitgevoerd. Ontbreekt een ELSE-instructie, dan gaat het programma naar de eerstvolgende uitvoerbare instructie. Direct voor THEN en ELSE mag een komma worden ingetikt.

Achter THEN en ELSE mag u een regelnummer plaatsen, maar ook een of meer instructies. Achter GOTO moet u altijd een regelnummer opgeven.

Als na een IF-opdracht het aantal THEN's niet gelijk is aan het aantal ELSE's dan zal elke ELSE-instructie door het programma worden gekoppeld aan de dichtstbijstaande THEN waarvoor geen ELSE is gevonden. Bijvoorbeeld:

```
IF A=B THEN IF B=C THEN PRINT "A=C" ELSE PRINT
"A<>C"
```

Als A ongelijk is aan B, dan zal de instructie PRINT "A<>C" niet worden uitgevoerd!

Als in de commando-modus achter een IF...THEN-opdracht een regelnummer wordt opgegeven, volgt de foutmelding "Ongedefinieerd regelnummer", behalve als er vooraf in de programma-modus een regel met het betreffende regelnummer is gebruikt.

Omdat IF...THEN...ELSE in feite één opdracht is, moet de gehele constructie in dezelfde programmaregel staan. Het is dus niet toegestaan ELSE in een andere programmaregel te zetten.

Nesten van IF-opdrachten:

Het is toegestaan IF...THEN...ELSE-opdrachten te "nesten", d.w.z. binnen een IF...THEN...ELSE-constructie een tweede soortgelijke constructie op te nemen. Het aantal geneste opdrachten wordt slechts beperkt door de maximale regellengte. Onderstaande programmaregel bevat een voorbeeld van geneste IF opdrachten:

```
100 IF X>Y THEN PRINT "X groter dan Y" ELSE IF X<Y
    THEN PRINT "X kleiner dan Y" ELSE PRINT "X
    gelijk aan Y"
```

Testen op gelijkheid:

Als u via IF-opdrachten wilt testen of getallen met decimalen aan elkaar gelijk zijn, bedenk dan dat de interne representatie van die getallen niet altijd gegarandeerd nauwkeurig is. Het is dan ook raadzaam niet alleen te testen op de exacte waarde van het getal zelf, maar te testen op het bereik waarbinnen de nauwkeurigheid van dat getal kan variëren.

Als u bijvoorbeeld wilt testen of de voor variabele A berekende waarde gelijk is aan 1.0, gebruikt u de volgende opdracht:

```
100 IF ABS(A-1.0) < 1.0E-6 THEN ...
```

Deze voorwaarde is waar als de waarde van A gelijk is aan 1.0 met een afwijking van minder dan 1.0E-6, ofwel 1 maal 10 tot de macht -6.

Voorbeeld 1:

```
100 IF (N<20) AND (N>10) THEN DB=1979-N: GOTO 300
110 PRINT "Getal ligt buiten bereik 10-20"
```

Dit programma test of de waarde van N groter is dan 10 en tevens kleiner dan 20. Als aan beide voorwaarden is voldaan, wordt DB berekend en gaat het programma verder met regel 300. In andere gevallen wordt regel 110 uitgevoerd.

Voorbeeld 2:

```
210 IF IOFLAG THEN PRINT A$ ELSE LPRINT A$
```

Deze opdracht stuurt A$ afhankelijk van de waarde van IOFLAG naar het beeldscherm of naar de printer. Als IOFLAG waar is (true, ongelijk aan 0) dan wordt A$ op het beeldscherm weergegeven. In andere gevallen is IOFLAG onwaar (false, gelijk aan 0) en wordt A$ afgedrukt.

Voorbeeld 3:

```
IF N THEN GET#1,N
```

Deze instructie leest record N in als N ongelijk is aan 0 (dus waar is).

INKEY$ variabele

Functie:

Opslaan van een via het toetsenbord ingetikt teken.

Syntaxis:
v$=INKEY$

Opmerkingen:
Als in de toetsenbordbuffer geen teken is opgeslagen, bevat de variabele INKEY$ een nul-string.

Als meerdere tekens in de buffer zijn opgeslagen, wordt alleen het eerste daarvan in INKEY$ opgeslagen. De tekenreeks is een of twee tekens lang.

Voor het weergeven van de uitgebreide codes uit appendix C worden twee tekenreeksen gebruikt. Het eerste teken van een code van twee tekens is een nul.

De toetscodes worden niet op het scherm afgebeeld. Alle tekens worden aan het programma doorgegeven, met uitzondering van:

CTRL-BREAK
CTRL-NUMLOCK
CTRL-ALT-DEL
CTRL-PRTSC
PRTSC

Voorbeeld:
```
10 CLS: PRINT "Druk op Enter-toets"
20 MAXTIJD%=1000
30 GOSUB 1000
40 IF GRENS% THEN PRINT "Te laat" ELSE PRINT
   "Keurig!"
50 PRINT TOETS$
60 END
.
.
.
1000 REM Subroutine max. toelaatbare tijd
1010 TOETS$=""
1020 FOR N%=1 TO MAXTIJD%
1030 A$=INKEY$
1040 IF ASC(A$)=13 THEN GRENS%=0: RETURN
1050 TOETS$=TOETS$+A$
1060 NEXT N%
1070 GRENS%=1:RETURN
```

Als u op ENTER drukt voordat de FOR...NEXT lus van 1 tot 1000 is doorlopen, verschijnt op het scherm "Keurig!". In andere gevallen was u te laat en wordt dat gemeld door de boodschap "Te laat".

Elke opdracht met de variabele INKEY$ controleert slechts één maal of een toets werd ingedrukt. Neem INKEY$ daarom op in een FOR...NEXT-lus om u voldoende tijd te gunnen voor het indrukken van een toets.

INP functie

Functie:
Byte inlezen vanaf de poort met het opgegeven nummer.

Syntaxis:
INP(*n*)

Opmerkingen:

Voor argument *n* moet een geldig poortnummer worden opgegeven dus een getal uit het bereik 0-65535.

INP is een van de communicatiemethoden tussen GW-BASIC en op de computer aangesloten randapparatuur.

INP voert de omgekeerde bewerking van de opdracht OUT uit.

Voorbeeld:

```
100 A=INP(56)
```

Na uitvoering van deze instructie bevat variabele A de waarde die aanwezig is op de poort met poortnummer 56. Dat is een decimale waarde van minimaal 0 en maximaal 255.

Dezelfde instructie luidt in assembleertaal:

```
MOV DX,56
IN AL,DX
```

INPUT opdracht

Functie:

Wachten op invoer via het toetsenbord en ingetikte informatie opslaan in variabelen.

Syntaxis:

INPUT [;] [*prompt;*] *variabele(n)*
INPUT [;] [*prompt,*] *variabele(n)*

Opmerkingen:

De *prompt* is een tekst die op het scherm aangeeft welke informatie het systeem verwacht.

Achter de prompt kunt u een of meer *variabelen* opgeven, waarin de gegevens worden opgeslagen die worden ingetikt.

Elke *prompt*-tekst moet tussen aanhalingstekens (") staan en worden gevolgd door een puntkomma of komma en minimaal één variabelenaam. Meerdere variabelenamen moeten onderling worden gescheiden door komma's.

De ingetikte informatie wordt in de variabele(n) opgeslagen. Er moeten evenveel gegevens worden ingetikt als er variabelenamen zijn opgenomen in de INPUT-opdracht.

De variabelenamen mogen verwijzen naar numerieke, tekenreeks- of matrix-variabelen. Elk ingetikt gegeven moet van hetzelfde type zijn als de variabele waarin het moet worden opgeslagen.

Als het aantal en/of het type gegevens niet overeenkomt met de variabelen, dan verschijnt de foutmelding "?Begin invoer opnieuw" (Fout, begin opnieuw). Ingetikte gegevens worden pas in de variabele(n) opgeslagen als alles correct is.

Door vervanging van de puntkomma door een komma zal achter de tekst geen vraagteken op het scherm worden afgebeeld:

```
INPUT "Tik geboortedatum in",G$
```

Als u vóór de prompt een puntkomma zet, negeert het systeem dat u na invoer van gegevens op <ENTER> drukt. Dit houdt in dat de inhoud van een volgende PRINT-opdracht direct achter de bij INPUT ingetikte gegevens wordt afgebeeld.

Zodra het programma een INPUT-opdracht tegenkomt, wordt eerst de prompt afgebeeld. Vervolgens wacht het programma tot u gegevens intikt. Als u in antwoord op een INPUT-opdracht tekst wilt intikken, is het niet noodzakelijk deze tussen aanhalingstekens te zetten, behalve wanneer in die tekst zelf komma's staan of aan het begin of einde spaties staan.

Zodra u na het intikken van gegevens op ENTER drukt, wordt de uitvoering van het programma hervat.

INPUT- en LINE INPUT-opdrachten bevatten een interne PRINT-instructie voor het weergeven van de prompt. Als u een prompt, tussen aanhalingstekens, aan de opdracht toevoegt, wordt deze dus automatisch afgebeeld. Zie ook de beschrijving van PRINT.

Het essentiële verschil tussen INPUT en LINE INPUT is dat LINE INPUT de mogelijkheid biedt tot het intikken van speciale tekens (zoals komma's) in een tekenreeks, zonder dat daarvoor dubbele aanhalingstekens zijn vereist. Bij INPUT zijn die wel noodzakelijk.

Voorbeeld 1:

```
10 INPUT X
20 PRINT "Het kwadraat van"X"is"X^2
30 END
RUN
?
```

De gebruiker tikt achter het vraagteken het getal 5 in en drukt op ENTER:

```
Het kwadraat van 5 is 25
Ok
```

Voorbeeld 2:

```
10 PI=3.14
20 INPUT "Hoe groot is de straal";R
30 A=PI*R^2
40 PRINT "De oppervlakte van de cirkel is";A
50 PRINT
60 GOTO 20
RUN
Hoe groot is de straal? 7.4
De oppervlakte van de cirkel is 171.9464
Hoe groot is de straal?
```

Het programma in voorbeeld 2 berekent op basis van de opgegeven straal de oppervlakte van een cirkel. Merk op dat INPUT op regel 20 automatisch de bijbehorende vraag om invoer van gegevens afbeeldt.

INPUT# opdracht

Functie:

Gegevens uit een sequentieel bestand inlezen en opslaan in programmavariabelen.

Syntaxis:

INPUT# *bestandsnummer*, *variabele(n)*

Opmerkingen:

Bij het argument *bestandsnummer* geeft u het nummer op dat bij het openen aan dat bestand werd toegekend.

Bij *variabele(n)* staan de variabelenamen waarin de ingelezen gegevens moeten worden opgeslagen.

Deze gegevens worden afgebeeld op de wijze die gebruikelijk is bij het intikken van gegevens na INPUT.

De variabele en de variabelenaam moeten van hetzelfde type zijn.

Na INPUT# wordt geen vraagteken afgebeeld, wat bij INPUT wel het geval is.

Numerieke waarden

Bij het inlezen van numerieke waarden worden spaties vóór of achter het getal genegeerd. Het eerste teken (dat geen spatie of line feed is) wordt beschouwd als begin van het getal. Het getal wordt als afgesloten beschouwd bij de eerstvolgende spatie, carriage return, line feed of komma.

Tekenreeksen

Als GW-BASIC in een sequentieel bestand zoekt naar tekenreeksen, negeert het aan die tekenreeks voorafgaande spaties en line feeds. Als een tekenreeks begint met dubbele aanhalingstekens ("), beschouwt GW-BASIC alle tekens tot het eerstvolgende aanhalingsteken als één tekenreeks. Dit betekent dat de tekenreeks zelf geen dubbele aanhalingstekens (") mag bevatten, omdat deze altijd als begin of eind van tekenreeksen worden beschouwd.

Als een tekenreeks niet begint met dubbele aanhalingstekens, beschouwt GW-BASIC de eerstvolgende komma, carriage return of line feed als einde van de tekenreeks. Bovendien kan een tekenreeks nooit langer zijn dan 255 tekens.

Als tijdens het inlezen van numerieke waarden of tekenreeksen het einde van een bestand wordt bereikt, wordt de procedure na het laatst ingelezen teken afgesloten.

INPUT# mag ook worden gebruikt voor het inlezen van gegevens uit onbepaald benaderbare bestanden.

INPUT$ functie

Functie:

Reeks tekens inlezen vanaf het toetsenbord of vanuit een bestand.

Syntaxis:

INPUT$(*x*[,[#]*bestandsnummer*])

Opmerkingen:

Als via deze functie gegevens vanaf het toetsenbord worden ingevoerd, verschijnen deze niet op het scherm. Behalve CTRL-BREAK worden alle gegevens naar het programma doorgestuurd. De genoemde toetsencombinatie beëindigt de functie INPUT$.

Voor het inlezen van gegevens die via de communicatiepoort worden aangeboden (COM-bestanden), verdient INPUT$ de voorkeur boven INPUT en LINE INPUT, omdat in de communicatie tussen computersystemen alle ASCII-tekens belangrijk kunnen zijn. INPUT is de slechtste keuze omdat het inlezen wordt afgebroken als de code voor een komma of een line feed wordt gelezen. Bij LINE INPUT geldt hetzelfde nadeel bij de code voor een carriage return.

Via INPUT$ worden alle ingelezen tekens verwerkt, bij INPUT$(x) het door x aangeduide aantal tekens.

Raadpleeg appendix F voor meer bijzonderheden.

Voorbeeld 1:

```
10 OPEN"I",1,"DATA"
20 IF EOF(1) THEN 50
30 PRINT HEX$(ASC(INPUT$(1,#1)));
40 GOTO 20
50 PRINT
60 END
```

Dit voorbeeldprogramma leest de gegevens uit het sequentiële bestand "DATA" en beeldt vervolgens op het scherm de hexadecimale versie van de ingelezen gegevens af.

Voorbeeld 2:

```
.
.
.
100 PRINT "Druk op d voor doorgaan of op s voor
stop"
110 X$=INPUT$(1)
120 IF X$="d" THEN 500
130 IF X$="s" THEN 700 ELSE 100
.
.
.
```

INSTR functie

Functie:

Aangeven op welke positie in een tekenreeks x$ een bepaalde tekenreeks y$ voor de eerste keer voorkomt.

Syntaxis:

INSTR([*n*,]x$,y$)

Opmerkingen:

Met het niet verplichte argument *n* kunt u aangeven vanaf welke tekenpositie x$ moet worden doorzocht. Als u geen waarde opgeeft, begint het zoeken naar y$ op de eerste positie van x$.

De waarde van *n* moet minimaal 1 zijn en mag niet groter zijn dan 255. Als *n* buiten dat bereik ligt, verschijnt de foutmelding "Onjuiste functieaanroep".

INSTR leidt tot het resultaat 0 als:

- n > LEN(x$)
- x$ een nul-string is (een lege tekenreeks)
- y$ niet wordt gevonden.

Als y$ leeg is, is het resultaat van INSTR gelijk aan *n*.

De argumenten x$ en y$ mogen tekenreeks-variabelen zijn, maar ook tekenreeks-expressies of letterlijke tekst tussen aanhalingstekens.

Voorbeeld:

```
10 X$="abcdebxyz"
20 Y$="b"
30 PRINT INSTR(X$,Y$);INSTR(4,X$,Y$)
2   6
Ok
```

Het programma doorzoekt de tekenreeks X$ en vindt dat Y$ (letter b) voor de eerste keer voorkomt op positie 2 in X$. Via de tweede INSTR-instructie wordt X$ doorzocht vanaf positie 4 (de letter d) en op positie 6 de tweede keer Y$ gevonden. Bij deze zoekprocedures worden de laatste drie tekens van X$ (xyz) genegeerd omdat regel 30 geen verdere opdrachten bevat.

INT functie

Functie:
Een getal of uitkomst van een expressie inkorten tot een geheel getal.

Syntaxis:
INT(*x*)

Opmerkingen:
Bij een negatieve waarde voor x leidt INT tot het direct aan *x* voorafgaande negatieve gehele getal.

Zie ook de beschrijving van FIX en CINT.

Voorbeeld 1:

```
PRINT INT (99.89)
99
Ok
```

Uit dit voorbeeld blijkt dat INT niet afrondt!

Voorbeeld 2:

```
PRINT INT(-12.11)
-13
Ok
```

IOCTL opdracht

Functie:
GW-BASIC in staat stellen een tekenreeks met control-codes naar een vooraf geopend stuurprogramma te sturen.

Syntaxis:
IOCTL[#]*bestandsnummer,tekenreeks*

Opmerkingen:
Bij *bestandsnummer* wordt het nummer opgegeven van het bestand dat voor het betreffende stuurprogramma is geopend.

Bij het argument *tekenreeks* moet een geldige tekenreeks-expressie worden opgegeven die de control-codes voor het stuurprogramma bevat.

IOCTL-instructies bestaan in het algemeen uit 2 of 3 tekens, gevolgd door een optioneel alfanumeriek argument. Een IOCTL-tekenreeks kan maximaal 255 tekens lang zijn. In de tekenreeks kunnen afzonderlijke commando's staan, van elkaar gescheiden door puntkomma's.

Voorbeeld 1:
```
OPEN "LPT1:" FOR OUTPUT AS #1
IOCTL #1, "PL66"
```

Stel dat u de printer driver LPT1: (line printer 1) hebt vervangen door een aangepaste printer driver en dat in die driver de paginalengte kan worden ingesteld (het aantal regels dat per pagina wordt afgedrukt, waarna een form feed wordt uitgevoerd om naar het volgende vel papier te gaan). De instructies uit bovenstaand voorbeeld kunnen dan de aangepaste printer driver LPT1: openen en de paginalengte instellen op 66 regels per pagina.

Voorbeeld 2:
```
OPEN "\DEV\LPT1:" FOR OUTPUT AS #1
IOCTL #1, "PL56"
```

Deze instructies openen driver LPT1: met een paginalengte van 56 regels.

IOCTL$ functie

Functie:
GW-BASIC in staat stellen een tekenreeks met control-codes te versturen vanuit een geopende driver.

Syntaxis:
IOCTL$[#](*bestandsnummer*)

Opmerkingen:
Bij het argument *bestandsnummer* geeft u het nummer op van het bestand dat voor de betreffende driver werd geopend.

IOCTL$ wordt in het algemeen gebruikt om te controleren of een IOCTL-instructie op correcte wijze werd uitgevoerd. Bovendien dient de functie voor het opvragen van informatie over apparatuur.

Voorbeeld:

```
10 'GW is een mogelijkheid voor
20 'opvragen van driverlengte
30 OPEN "\DEV\LPRT" AS #1
40 IOCTL#1,"GW"
50 REM Opslaan in LANG
60 LANG=VAL(IOCTL$(#1))
```

KEY opdracht

Functie:

Maximaal 15 tekens tegelijk invoeren, via herprogrammeren van de functietoetsen van GW-BASIC.

Syntaxis:

KEY *toetsnummer*, *tekenreeks-expressie*
KEY *n*,CHR$(*hexcode*)+CHR$(*scancode*)
KEY ON
KEY OFF
KEY LIST

Opmerkingen:

Bij *toetsnummer* geeft u het nummer van de toets op (uit het bereik 1 t/m 20) die u wilt programmeren.

Het argument *tekenreeks-expressie* mag uit maximaal 15 tekens bestaan, de resterende tekens worden genegeerd. Constanten moeten tussen dubbele aanhalingstekens staan.

Bij *scancode* geeft u aan welke toets u wilt detecteren. Appendix H geeft een overzicht van de scan-codes voor alle toetsen.

Het argument *hexcode* verwijst naar onderstaande hexadecimale toets-codes:

Toets	Hexcode
EXTENDED*)	&H80
CAPS LOCK	&H40
NUM LOCK	&H20
ALT	&H08
CTRL	&H04
SHIFT	&H01, &H02, &H03

*) Met "extended" worden de toetsen bedoeld die op een uitgebreid toetsenbord (101/102 toetsen) zijn toegevoegd. Met name de cursor-toetsen, INS, DEL, HOME, END, PAGE UP en PAGE DN.

Hexcodes mogen bij elkaar worden geteld. Zo duidt &H03 op beide SHIFT-toetsen.

Normaal gesproken zijn onder de functietoetsen onderstaande tekenreeksen opgeslagen:

F1	LIST	F2	RUN<-
F3	LOAD"	F4	SAVE"
F5	CONT<-	F6	,"LPT1:"<-
F7	TRON<-	F8	TROFF<-
F9	KEY	F10	SCREEN 0,0,0<-

Attentie:
Het pijlsymbool <- duidt aan dat u na de betreffende functietoets te hebben ingedrukt niet op ENTER hoeft te drukken.

Elk van deze 10 functietoetsen kan opnieuw worden geprogrammeerd, zodat er een andere tekenreeks in wordt opgeslagen. De instructie:

```
KEY toetsnummer, "tekenreeks-expressie"
```

zorgt dat de opgegeven *tekenreeks-expressie* in de toets met het opgegeven toetsnummer wordt opgeslagen.

Via de instructie KEY LIST krijgt u op het scherm een overzicht welke tekenreeks in elk van de functietoetsen is opgeslagen.

Via KEY ON worden op schermregel 25 (onderin het scherm) van elke opgeslagen tekenreeks de eerste 6 tekens afgebeeld. Als de schermbreedte op 40 is ingesteld, worden 5 van de 10 toetsen afgebeeld; bij een schermbreedte van 80 tekens worden alle 10 de toetsen afgebeeld.

Via KEY OFF wordt de informatie over de functietoetsen weer van schermregel 25 verwijderd, waarna die regel weer voor andere doeleinden kan worden gebruikt. De instructie KEY OFF heft de geprogrammeerde tekenreeksen van de functietoetsen niet op.

Als u een toetsnummer opgeeft buiten het bereik 1 t/m 10, of 15 t/m 20, dan volgt de foutmelding "Onjuiste functieaanroep". De tot dan toe geprogrammeerde tekenreeksen blijven in dat geval ongewijzigd.

Als u onder een toets een nul-string (tekenreeks met een lengte 0) opslaat, fungeert de toets niet langer als geprogrammeerde toets.

Als een toets opnieuw wordt geprogrammeerd, zal één keer activeren van de functie INKEY$ één teken uit de geprogrammeerde tekenreeks opslaan. Als de toets niet langer als functietoets fungeert, geeft INKEY$ een tekenreeks van twee tekens, waarvan het eerste binair 0 is en het tweede de scancode van de betreffende toets bevat.

Voorbeeld 1:
```
10 KEY 1,"MENU"+CHR$(13)
```

Deze instructie laat een menu verschijnen zodra functietoets 1 wordt ingedrukt.

Voorbeeld 2:
```
1 KEY OFF
```

Wist de regel met informatie over de functietoetsen van het scherm.

Voorbeeld 3:
```
10 DATA KEY1, KEY2, KEY3, KEY4, KEY5
20 FOR N=1 TO 5:READ FUNCTOETS$(N)
30 KEY N, FUNCTOETS$(N)
40 NEXT N
50 KEY ON
```

Dit programma geeft op schermregel 25 informatie over de tekenreeksen die in functietoetsen zijn opgeslagen.

Voorbeeld 4:

```
20 KEY 1,""
```

Functietoets 1 fungeert niet langer als functietoets, omdat er een nul-string in wordt opgeslagen.

KEY(n) opdracht

Functie:

In- en uitschakelen van het controleren of bepaalde toetsen worden ingedrukt tijdens een GW-BASIC programma.

Syntaxis:

KEY(*n*) ON
KEY(*n*) OFF
KEY(*n*) STOP

Opmerkingen:

Het argument *n* is een nummer uit het bereik 1 t/m 20. Het geeft aan van welke toets moet worden gedetecteerd dat hij wordt ingedrukt. De toetsen zijn als volgt genummerd:

Toetsnummer	Toets
1 - 10	Functietoetsen F1 t/m F10
11	Cursor omhoog
12	Cursor naar links
13	Cursor naar rechts
14	Cursor omlaag
15 - 20	Toetsen die als volgt worden gedefinieerd (zie de KEY opdracht): KEY n,CHR$(*hexcode*) + CHR$(*scancode*)

Voor het activeren van de detectie moet een KEY(*n*) ON-opdracht worden uitgevoerd. Als een KEY(*n*) ON-opdracht wordt geactiveerd, controleert GW-BASIC elke volgende opdracht om na te gaan of de aangeduide toets is ingedrukt. Als dat het geval is, voert GW- BASIC een GOSUB-instructie uit naar het regelnummer dat in de ON KEY(*n*)-opdracht is opgenomen. Een KEY(*n*)-opdracht moet dus worden voorafgegaan door een ON KEY(*n*)-opdracht.

Wanneer een KEY(*n*) OFF-instructie wordt uitgevoerd, wordt niet langer gecontroleerd of de betreffende toets wordt ingedrukt.

Na een KEY(*n*) STOP-opdracht reageert het programma niet meer op het indrukken van een geprogrammeerde toets, maar onthoudt wel dat die is ingedrukt. Bij de eerstvolgende KEY(*n*) ON-instructie voert het programma dan alsnog de geprogrammeerde instructie uit.

Raadpleeg de opdracht ON KEY(*n*) voor meer bijzonderheden.

KILL commando

Functie:

Wissen van een schijfbestand.

Syntaxis:
KILL *bestandsnaam*

Opmerkingen:
Bij *bestandsnaam* kunt u een programmanaam opgeven, maar ook sequentiële of onbepaald benaderbare bestanden.

Attentie:
Bij toepassing van de KILL-opdracht moet ook het bestandstype van de bestandsnaam worden opgegeven. Bestanden die met GW-BASIC zijn opgeslagen, hebben het bestandstype *.bas*.

Als u probeert met KILL een op dat moment geopend bestand te wissen, leidt dat tot de foutmelding "Bestand reeds open".

Voorbeeld 1:
```
200 KILL "DATA1.BAS"
```

Deze programmaregel wist het GW-BASIC bestand "DATA1", zodat de daardoor in beslag genomen ruimte vrij komt voor andere bestanden.

Voorbeeld 2:
```
KILL "HANS\APP\ELEK.BAS"
```

Deze instructie wist het bestand *elek.bas* uit de subdirectory *app*.

LEFT$ functie

Functie:
Een tekenreeks vormen, bestaande uit het opgegeven aantal tekens, geteld vanaf het begin van een andere tekenreeks.

Syntaxis:
LEFT$(x$,*n*)

Opmerkingen:
Argument *n* moet een waarde hebben uit het bereik 0-255. Als *n* groter is dan de lengte van x$ (=LEN(x$)) dan wordt x$ in zijn geheel opgeslagen in de nieuwe tekenreeks. Als *n* gelijk is aan 0, dan resulteert LEFT$ in een nul-string (lengte 0, lege tekenreeks). Zie ook de beschrijving van MID$ en RIGHT$.

Voorbeeld:
```
10 A$="Basje"
20 B$=LEFT$(A$,3)
30 PRINT B$
RUN
Bas
Ok
```

De eerste 3 tekens uit A$ worden opgeslagen in B$ en op het scherm afgebeeld.

LEN functie

Functie:
Aangeven hoe lang een tekenreeks is.

Syntaxis:
LEN(x$)

Opmerkingen:
Bij het bepalen van de lengte van een tekenreeks worden ook spaties en niet op het scherm verschijnende speciale tekens meegeteld. Voor x$ kan een willekeurige tekenreeks worden opgegeven.

Voorbeeld:
```
10 X$="Woonplaats: Eindhoven"
20 PRINT LEN(X$)
RUN
21
Ok
```

Merk op dat bij het bepalen van de lengte van X$ ook de dubbele punt en de spaties worden meegeteld.

LET opdracht

Functie:
De waarde van een expressie opslaan in een variabele.

Syntaxis:
[LET] *variabelenaam=expressie*

Opmerkingen:
De opdracht LET is niet verplicht. U kunt hetzelfde effect bereiken door alleen de variabelenaam en de expressie in te tikken, met daartussen het is-gelijk teken (A=3). De LET-opdracht wordt zelden gebruikt en is hier slechts beschreven omdat het in sommige oude versies van BASIC wel verplicht is.

Vergeet niet dat het type van de variabele en het resultaat van de expressie of de numerieke waarde gelijk moeten zijn. Als dat niet het geval is, verschijnt de foutmelding "Onjuist type".

Voorbeeld 1:
```
110 LET D=12
120 LET E=12^2
130 LET F=12^4
140 LET SOM =D+E+F
.
.
.
```

Voorbeeld 2:
```
110 D=12
120 E=12^2
130 F=12^4
140 SOM =D+E+F
.
.
.
```

Het resultaat van voorbeeld 1 is gelijk aan dat van voorbeeld 2.

LINE opdracht

Functie:
Lijnen en rechthoeken op het scherm tekenen.

Syntaxis:
LINE [(*x1,y1*)]-(*x2,y2*)[,[*kenmerk*][,B[F]][,*stijl*]]

Opmerkingen:
De argumenten (*x1,y1*) en (*x2,y2*) bepalen de uiteinden van een lijn.

De resolutie wordt gekozen met een SCREEN-opdracht.

Het argument *kenmerk* bepaalt de kleur en de intensiteit van de afgebeelde beeldpunten (zie ook de opdrachten COLOR en PALETTE).

Door argument B (van "Box") toe te voegen, verschijnt een rechthoek op het scherm met hoekpunten op de coördinaten (*x1,y1*) en (*x2,y2*).

Als u BF (van "Box Filled") opgeeft, wordt een rechthoek getekend (zoals ook bij ,B) die wordt opgevuld met puntjes.

Attentie:
Als u voor het argument *kenmerk* niets opgeeft, moet u vóór B of BF twee komma's intikken.

LINE bevat ook het argument *stijl*. Dat is een masker (integer, 16 bits) dat wordt gebruikt voor het weergeven van beeldpunten op het scherm. Dit wordt *line-styling* genoemd. Steeds wanneer LINE een punt op het scherm opslaat, gebruikt het de huidige bit voor de stijl. Is de waarde van die bit 0, dan wordt niets opgeslagen en bij een bit-waarde 1 wordt normaal opgeslagen. Na elk punt wordt de eerstvolgende bit in het argument *stijl* gekozen.

Omdat een bit-waarde van 0 de oude inhoud niet wist, wilt u misschien een achtergrondlijn laten tekenen over een bestaande lijn met een bepaalde *stijl*, zodat u een gewenste achtergrond verkrijgt.

Het argument *stijl* wordt gebruikt voor normale lijnen en rechthoeken, maar is niet toegestaan bij het tekenen van ingekleurde rechthoeken.

Als u probeert de parameter BF te combineren met *stijl*, leidt dat tot een syntaxis-fout.

Als u voor argumenten in een LINE-opdracht waarden opgeeft die niet zijn toegestaan, zullen de coördinaten die fout worden opgegeven niet op het scherm zichtbaar worden. Dit wordt *line clipping* genoemd. Er wordt als het ware een deel van de tekening afgesneden.

In de syntaxis van de LINE-opdracht wordt geen melding gemaakt van de coördinaatvorm "STEP (*x offset, y offset*)". Deze vorm kan echter voor elke coördinaat worden gebruikt.

Als in een LINE-opdracht voor de tweede coördinaat een relatieve waarde wordt gebruikt, dan wordt die bepaald op basis van de eerste coördinaat.

Na uitvoering van een LINE-opdracht is (x2,y2) de huidige grafische positie.

De eenvoudigste vorm van een LINE-opdracht is:

LINE -(*xz,yz*)

Deze instructie tekent een lijn in de voor de voorgrond geactiveerde kleur, vanaf de huidige grafische positie naar het punt met de coördinaten (*xz,yz*).

Voorbeelden:

```
LINE (0,100)-(639,100)
```

Een horizontale lijn deelt het beeldscherm op in een boven- en onderhelft (SCREEN 2).

```
LINE (160,0)-(160,199)
```

Een verticale lijn deelt het scherm op in een linker- en rechterhelft (SCREEN 1). Deelt het scherm op in een kwart en drie kwart van de breedte (SCREEN 2).

```
LINE (0,0)-(319,199)
```

Diagonale lijn van linksboven naar rechtsonder (SCREEN 1). Diagonale lijn van linksboven naar het middelpunt van de onderrand van het scherm (SCREEN 2).

```
LINE (10,10)-(20,20),2
```

Een lijn in kleur 2 (SCREEN 1). Zie ook de COLOR-opdracht.

```
10 CLS
20 LINE -(RND*319,RND*199),RND*4
30 GOTO 20
```

Dit programma gaat onafgebroken door met het tekenen van lijnen en gebruikt daarbij willekeurige kenmerken.

```
10 FOR X=0 TO 319
20 LINE (X,0)-(X,199),X AND 1
30 NEXT
```

Tekent een afwisselend patroon.

```
10 CLS
20 LINE -(RND*639,RND*199),RND*2,BF: GOTO 20
```

Blijft onafgebroken lijnen tekenen over het gehele scherm.

```
LINE (0,0)-(100,175),,B
```

Tekent een vierkant linksboven in het scherm.

```
LINE (0,0)-(100,175),,BF
```

Tekent hetzelfde vierkant, maar ingekleurd (SCREEN 1).

```
LINE (0,0)-(100,350),,B
```

Tekent hetzelfde vierkant als SCREEN 2 is geactiveerd.

```
400 SCREEN 1
410 LINE (160,100)-(160,199),,,&HCCCC
```

Tekent een verticale stippellijn over het midden van het scherm (SCREEN 1).

```
220 SCREEN 2
230 LINE (300,100)-(400,50),,B,&HAAAA
```

Tekent een rechthoek met behulp van een stippellijn (SCREEN 2).

```
LINE (0,0)-(160,1000),3,,&HFF00
```

Tekent een stippellijn vanuit de linkerbovenhoek naar het centrum van het scherm.

LINE INPUT opdracht

Functie:

Wachten op invoer via het toetsenbord van een hele regel (max. 255 tekens) en de ingetikte informatie opslaan in een tekenreeks-variabele, waarbij scheidingstekens worden genegeerd.

Syntaxis:

LINE INPUT [;][*prompt;*]*tekenreeks-variabele*

Opmerkingen:

De *prompt* is een letterlijk op het scherm weer te geven tekst, die u duidelijk maakt dat u informatie kunt intikken. Het gebruik van een *prompt* is niet verplicht.

LINE INPUT plaatst geen vraagteken op het scherm, behalve als u de *prompt* zelf afsluit met een vraagteken.

Alle tekens die na de prompt worden ingetikt, worden in de *tekenreeks-variabele* opgeslagen, tot u op ENTER drukt. Als u begint met spaties, worden die genegeerd.

LINE INPUT vertoont sterke overeenkomst met INPUT. Maar een belangrijk verschil is dat LINE INPUT u toestaat speciale tekens in te voeren (zoals komma's), wat bij INPUT niet mogelijk is.

Als in de ingevoerde gegevens een line feed en carriage return (in deze volgorde!) voorkomen, worden beide tekens geaccepteerd en afgebeeld. De invoer van gegevens wordt niet afgebroken.

Als direct achter LINE INPUT een puntkomma staat, zal de cursor niet naar de volgende regel gaan als u op ENTER drukt.

U kunt het intikken van gegevens bij een LINE INPUT-opdracht afbreken door op CTRL-BREAK te drukken. GW-BASIC gaat dan naar de commando-modus en de prompt OK verschijnt.

Als u CONT intikt wordt de uitvoering van het programma hervat op de regel met de LINE INPUT-opdracht.

Voorbeeld:

```
100 LINE INPUT A$
```

De uitvoering van het programma stopt op regel 100. Alle tekens die u vervolgens intikt, worden in A$ opgeslagen, tot u op ENTER drukt of op een van de volgende toetscombinaties: CTRL-M, CTRL-C of CTRL-BREAK.

LINE INPUT# opdracht

Functie:

Inlezen van hele regel (max. 255 tekens), zonder scheidingstekens, vanuit een sequentieel schijfbestand en de ingelezen gegevens opslaan in een tekenreeks-variabele.

Syntaxis:

LINE INPUT# *bestandsnummer*, *tekenreeks-variabele*

Opmerkingen:

Het argument *bestandsnummer* moet het nummer zijn dat bij het openen van het bestand werd toegekend.

Bij *tekenreeks-variabele* moet de naam worden opgegeven van de variabele waarin de gegevens moeten worden opgeslagen.

LINE INPUT# leest alle tekens uit het sequentiële bestand tot aan de eerste carriage return. Een line feed, gevolgd door een carriage return (alleen in deze volgorde!) wordt geaccepteerd.

LINE INPUT# is vooral nuttig als elke regel van een gegevensbestand is onderverdeeld in afzonderlijke velden. Bovendien is deze opdracht zeer geschikt voor het inlezen van een ander GW-BASIC programma dat als ASCII-bestand is opgeslagen.

Voorbeeld:

```
10 OPEN "O",1,"INFO"
20 LINE INPUT "Klanteninformatie? ";K$
30 PRINT #1, K$
40 CLOSE 1
50 OPEN "I",1,"INFO"
60 LINE INPUT #1, K$
70 PRINT K$
80 CLOSE 1
RUN
Klanteninformatie?
```

Als u nu intikt:

```
De heer Michels,  5500 AA Tilburg
```

vervolgt het programma met:

```
De heer Michels,  5500 AA Tilburg
Ok
```

LIST commando

Functie:

Alle programmaregels of een deel daarvan op het scherm afbeelden, via de printer afdrukken of in een bestand opslaan.

Syntaxis:
LIST [*regelnummer*][-*regelnummer*][,*bestandsnaam*]
LIST [*regelnummer*-][*bestandsnaam*]

Opmerkingen:
Bij *regelnummer* moet u een regelnummer opgeven tussen 0 en 65529.

Als u geen *bestandsnaam* opgeeft, worden de opgegeven regels naar het beeldscherm gestuurd.

U kunt ook een bepaald deel van het programma kiezen door twee *regelnummers* op te geven, gescheiden door een koppelteken (-). Als u één regelnummer opgeeft, gevolgd door het koppelteken, dan worden alle regels afgebeeld vanaf de opgegeven regel. Als u één *regelnummer* opgeeft, voorafgegaan door het koppelteken, dan worden alle regels afgebeeld vanaf het begin van het programma tot en met de opgegeven regel.

Door na LIST alleen een punt (.) in te tikken, wordt de programmaregel afgebeeld die als laatste op het scherm is verschenen.

U kunt het afbeelden van programmaregels (ook wel *listing* genoemd) afbreken door op CTRL-BREAK te drukken.

Voorbeeld 1:
```
LIST
```

Geeft een listing van alle programmaregels.

Voorbeeld 2:
```
LIST -20
```

Overzicht van de regels 1 t/m 20.

Voorbeeld 3:
```
LIST 10-20
```

De regels met een regelnummer uit het bereik 10 t/m 20.

Voorbeeld 4:
```
LIST 20-
```

Listing van de regels met een regelnummer van 20 en hoger.

LLIST commando

Functie:
Via de printer afdrukken van alle programmaregels of een deel daarvan.

Syntaxis:
LLIST [*regelnummer*][-*regelnummer*]
LLIST [*regelnummer*-]

Opmerkingen:
Na de uitvoering van een LLIST-commando keert GW-BASIC altijd terug naar de commando-modus. Het kiezen van een bepaald deel van de programmaregels verloopt zoals dat bij LIST werd beschreven. Zie de voorbeelden bij LIST.

LOAD commando

Functie:
Een bestand vanaf diskette in het geheugen laden.

Syntaxis:
LOAD *bestandsnaam*[,r]

Opmerkingen:
Bij *bestandsnaam* geeft u de naam op die werd toegekend toen het bestand werd opgeslagen. Als u geen bestandstype opgeeft, kiest GW-BASIC automatisch het bestandstype *.bas* voor een BASIC programma.

LOAD sluit alle geopende bestanden, wist alle variabelen en programmaregels die zich in het geheugen bevinden en laadt daarna het opgegeven bestand.

Als u de toevoeging ,r gebruikt, wordt het programma na het laden automatisch uitgevoerd. Bovendien worden in dat geval de geopende bestanden niet eerst gesloten. Door de toevoeging ,r kunt u verschillende programma's (of delen van één programma) aan elkaar koppelen. U kunt dan informatie tussen de verschillende programma's uitwisselen met behulp van gegevensbestanden op diskette.

Voorbeeld:
```
LOAD "STATPROG",R
```

Het BASIC programma STATPROG.BAS wordt geladen en vervolgens automatisch opgestart. Alle geopende bestanden blijven open en de waarden van variabelen in het huidige programma blijven ongewijzigd.

LOC functie

Functie:
De huidige positie in het bestand aangeven.

Syntaxis:
LOC (*bestandsnummer*)

Opmerkingen:
Bij *bestandsnummer* geeft u het nummer op dat bij het openen aan dat bestand werd toegekend.

Als u via een communicatiepoort gegevens met een bestand uitwisselt, geeft LOC het aantal tekens aan dat in de invoerbuffer wacht op verwerking. De standaard bufferruimte is 256 tekens. U kunt die waarde wijzigen door op de commandoregel van GW-BASIC de optie /c: te gebruiken. Als de buffer meer dan 255 tekens bevat, geeft LOC toch de waarde 255. Omdat een tekenreeks maximaal 255 tekens lang mag zijn, vereenvoudigt deze praktische bovengrens van de LOC-waarde het testen van de tekenreeks-lengte voordat u gegevens in die tekenreeks opslaat. Als de buffer minder dan 255 tekens bevat, geeft LOC het juiste aantal tekens in de buffer.

Bij communicatie met onbepaald benaderbare bestanden geeft LOC het nummer van het record dat als laatste werd ingelezen (GET) of het laatste record waarin gegevens werden weggeschreven (PUT).

Bij sequentiële bestanden geeft LOC het aantal blokken (elk 128 bytes) dat sinds het openen van dat bestand werd verwerkt in lees- of schrijf-operaties. Als het bestand werd geopend voor invoer, leest GW-BASIC in eerste instantie de eerste sector van dat bestand. In dit geval geeft LOC eerst het cijfer 1, alvorens invoer van gegevens mogelijk wordt.

Als het bestand werd geopend, maar geen gegevenstransport van of naar diskette plaatsvond, geeft LOC de waarde 0.

Voorbeeld:

```
200 IF LOC(1)#50 THEN STOP
```

Het programma stopt nadat 51 records zijn ingelezen of weggeschreven.

LOCATE opdracht

Functie:

Verplaatsen van de cursor naar een bepaalde positie op het beeldscherm. Optionele parameters maken de cursor knipperend en bepalen de afmeting en vorm van de cursor; het laatste via het opgeven van rasterregels voor het begin en einde van de cursor. Een rasterregel is de verticale of horizontale afstand tussen twee bij elkaar gelegen adresseerbare posities op het scherm.

Syntaxis:

LOCATE [*rij*][,[*kol*][,[*cursor*][,[*start*][,*stop*]]]]

Opmerkingen:

Het argument *rij* is het nummer van de schermregel. *Rij* moet een getal zijn tussen 1 en 25.

Het argument *kol* verwijst naar de kolom op het scherm (de positie op de schermregel). Het moet een getal zijn tussen 1 en 40, of tussen 1 en 80, afhankelijk van de ingestelde schermbreedte.

De logische waarde van *cursor* bepaalt of de cursor zichtbaar is. Bij de waarde 0 is de cursor onzichtbaar, bij andere waarden is hij zichtbaar.

Het argument *start* bepaalt de eerste regel en het argument *stop* de laatste regel van een raster. Beide kunnen een waarde hebben tussen 0 en 31. Samen bepalen zij afmeting en vorm van de cursor.

Als de cursor naar de opgegeven positie wordt verplaatst, zal de inhoud van volgende PRINT-opdrachten vanaf de nieuwe positie worden weergegeven. De opdracht LOCATE kan ook worden gebruikt om knipperen te activeren en uit te schakelen en om de afmeting van de cursor aan te passen.

Als voor een argument een waarde buiten het geldige bereik wordt opgegeven, leidt dat tot de foutmelding "Onjuiste functieaanroep" en worden de bestaande waarden niet aangepast.

Als u voor een argument geen nieuwe waarde wilt opgeven, moet u voor dat argument een komma in de opdracht plaatsen. Dit geldt niet als de argumenten die u overslaat het einde van de opdracht vormen.

Als u een waarde voor *start* opgeeft zonder een waarde voor *stop* op te geven, dan geldt de *start*-waarde tevens voor het argument *stop*.

Voorbeeld 1:

```
10 LOCATE 1,1
```

Verplaatst de cursor naar de linkerbovenhoek, de positie "home".

Voorbeeld 2:

```
20 LOCATE ,,1
```

Maakt de cursor zichtbaar. De positie van de cursor blijft ongewijzigd, want de twee komma's duiden aan dat de eerste twee argumenten worden overgeslagen.

Voorbeeld 3:

```
30 LOCATE ,,,7
```

De cursor blijft op dezelfde positie en blijft zichtbaar. De afmeting en vorm van de cursor veranderen: het wordt een streepje onder de tekenpositie, omdat de start-waarde "regel 7" ook als stop-waarde geldt.

Voorbeeld 4:

```
40 LOCATE 5,1,1,0,7
```

De cursor gaat naar schermregel 5, kolom 1 en wordt zichtbaar gemaakt. De afmeting van de cursor beslaat nu een gehele tekenpositie, lopend van regel 0 t/m 7 in het raster.

LOCK opdrachten

Functie:

Een bestand of een deel daarvan ontoegankelijk maken. Deze opdracht wordt vaak gebruikt in configuraties met meerdere computersystemen, ook wel *netwerken* genoemd.

Syntaxis:

LOCK [#]*n*[,[*record-nummer*][**TO** *record-nummer*]]

Opmerkingen:

Voor *n* geeft u het bestandsnummer op waarmee het bestand in het programma werd aangeduid.

Als u één bepaald record ontoegankelijk wilt maken, geeft u het nummer van dat record op. Als u een aaneengesloten reeks records ontoegankelijk wilt maken, geeft u het nummer van het eerste en laatste record van die reeks op.

De minimale waarde voor een *record-nummer* is 1. De maximale waarde is 2^{32}-1. De maximale lengte van een record is 32767 bytes.

Als u een reeks records opgeeft, mag het eerste *record-nummer* nooit groter zijn dan het tweede.

Als u geen nummer voor het eerste record in de reeks opgeeft, wordt daarvoor automatisch *record-nummer* 1 gekozen. Als u geen nummer voor het laatste record opgeeft, wordt alleen het opgegeven record ontoegankelijk.

Voorbeelden van correcte LOCK-opdrachten:

LOCK #*n*	maakt het gehele bestand *n* ontoegankelijk
LOCK #*n*,*X*	maakt record *X* ontoegankelijk
LOCK #*n*, TO *Y*	maakt de records *1* t/m *Y* ontoegankelijk
LOCK #*n*,*X* TO *Y*	maakt de records *X* t/m *Y* ontoegankelijk

Bij onbepaald benaderbare bestanden (random access files) kan het gehele geopende bestand of een reeks records in dat bestand ontoegankelijk worden gemaakt. Op die wijze worden die records ook afgesloten voor andere procedures waarmee het bestand eerder werd geopend.

Bij sequentiële bestanden wordt via LOCK het gehele bestand ontoegankelijk, ook als u slechts een reeks records zou opgeven. Dit is geen fout; de opgegeven record-reeks wordt bewust genegeerd.

Alvorens schrijf- of leesbewerkingen met een bestand uit te voeren, moet een LOCK-opdracht worden uitgevoerd.

Een via LOCK afgesloten bestand of reeks records moet weer toegankelijk worden gemaakt met UNLOCK, voordat u het bestand sluit (CLOSE). Als u dit vergeet, loopt u met name in een netwerk het risico dat het bestand later ontoegankelijk blijft.

In het algemeen geldt dat bestanden of records slechts voor korte tijd ontoegankelijk behoeven te zijn. Daarom volgen LOCK en UNLOCK opdrachten vaak kort na elkaar.

Een voorbeeld van een correcte combinatie van LOCK en UNLOCK opdrachten:

```
LOCK #1, 1 TO 4
LOCK #1, 5 TO 8
UNLOCK #1, 1 TO 4
UNLOCK #1, 5 TO 8
```

Een voorbeeld van foute combinatie van LOCK en UNLOCK:

```
LOCK #1, 1 TO 4
LOCK #1, 5 TO 8
UNLOCK #1, 1 TO 8
```

LOF functie

Functie:
De lengte van een bestand aangeven, uitgedrukt in het aantal bytes.

Syntaxis:
LOF (*bestandsnummer*)

Opmerkingen:
Bij *bestandsnummer* geeft u het nummer op dat werd toegekend toen het bestand werd geopend.

Bij communicatie-bestanden geeft LOF aan hoeveel ruimte vrij is in de invoerbuffers.

Voorbeeld:

```
10 OPEN "R",1,"FILE3.GRT"
20 GET #1,LOF(1)/128
 .
 .
 .
```

Het laatste record uit het onbepaald benaderbaar bestand *file3.grt* wordt opgehaald. De instructies gaan uit van een bestand met de standaard record-lengte van 128 bytes.

LOG functie

Functie:
Berekening van de natuurlijke logaritme.

Syntaxis:
LOG(x)

Opmerkingen:
De waarde voor x moet altijd groter zijn dan 0.

LOG(x) wordt berekend met enkele precisie, behalve wanneer GW-BASIC werd opgestart met /d.

Voorbeeld 1:
```
PRINT LOG(2)
.6931471
```

Voorbeeld 2:
```
PRINT LOG(1)
0
```

LPOS functie

Functie:
Aangeven van de huidige positie van de print-kop in de printer-buffer.

Syntaxis:
LPOS(x)

Opmerkingen:
LPOS geeft niet per definitie de exacte fysieke positie van de print-kop aan.

Argument x is een zogenaamd "dummy-argument".

Als de maximale regellengte van de printer kleiner is dan 132 tekens, kan het voorkomen dat de printer zelf de regel eerder afbreekt met interne line feeds die niet in de printer-buffer van de computer worden opgeslagen. In dat geval bestaat de kans dat de door LPOS aangegeven waarde onjuist is. LPOS(x) baseert de positie van de kop namelijk op het aantal afgedrukte tekens na de meest recente line feed.

Voorbeeld:
```
100 IF LPOS(X)=60 THEN LPRINT CHR$(13)
```

Via deze programmaregel gaat de printkop naar de volgende regel (carriage return) zodra 60 tekens op die regel zijn afgedrukt.

LPRINT en LPRINT USING opdrachten

Functie:
Gegevens afdrukken.

Syntaxis:
LPRINT [*lijst expressies*][;]
LPRINT USING *tekenreeks*; *lijst expressies*[;]

Opmerkingen:
De *lijst expressies* bestaat uit tekenreeksen of numerieke expressies, onderling gescheiden door puntkomma's.

Het argument *tekenreeks* is een letterlijke tekst tussen aanhalingstekens ("), ofwel een variabele met speciale format-tekens. Deze tekens bepalen het veld en het format van afgedrukte tekst en getallen.

LPRINT en LPRINT USING voor afdrukken via de printer komen overeen met de opdrachten voor weergave op het beeldscherm: PRINT en PRINT USING. Bij de beschrijving van de laatstgenoemde opdrachten vindt u meer bijzonderheden over de invulling van de argumenten.

LPRINT en LPRINT USING veronderstellen dat u een printer hebt aangesloten die 80 tekens per regel afdrukt. Met behulp van de opdracht WIDTH kunt u zelf de regellengte aanpassen.

LSET en RSET opdrachten

Functie:
Gegevens vanuit het werkgeheugen naar een bestandsbuffer sturen of gegevens links of rechts aansluitend maken als voorbereiding voor een PUT-opdracht.

Syntaxis:
LSET *tekenreeks-variabele=tekenreeks-expressie*
RSET *tekenreeks-variabele=tekenreeks-expressie*

Opmerkingen:
Als voor de *tekenreeks-expressie* minder bytes nodig zijn dan via een FIELD-opdracht voor de *tekenreeks-variabele* werden gereserveerd, dan zorgt LSET dat de tekenreeks links-aangesloten in het veld staat en RSET plaatst de tekenreeks rechts-aangesloten in het veld. De resterende posities worden opgevuld met spaties.

Als de tekenreeks langer is dan de via FIELD gereserveerde ruimte, dan worden de achterste tekens verwijderd.

Raadpleeg de beschrijving van de functies MKI$, MKS$ en MKD$. Daarmee zet u numerieke waarden om in tekenreeksen voordat u de opdrachten LSET of RSET gebruikt.

LSET en RSET kunnen ook worden toegepast om tekenreeks-variabelen waarvoor geen FIELD opdracht is gebruikt links- of rechts- aangesloten in een bepaald veld te plaatsen.

Voorbeeld:

```
110 A$=SPACE$(20)
120 RSET A$=N$
```

Met behulp van deze twee programmaregels wordt de tekenreeks N$ rechts-aangesloten in een veld van 20 tekens. Dergelijke instructies zijn nuttig om de layout voor afgedrukte gegevens te bepalen.

MERGE commando

Functie:

Koppelen van ASCII programmabestanden aan het programma dat al in het geheugen is opgeslagen.

Syntaxis:

MERGE *bestandsnaam*

Opmerkingen:

Bij bestandsnaam geeft u een geldige bestandsnaam op. Als u geen bestandstype opgeeft, kiest GW-BASIC automatisch voor de bestandstype *.bas* voor een BASIC programma.

MERGE zoekt op de diskette naar de opgegeven bestandsnaam. Als deze wordt gevonden, worden de programmaregels in dat bestand samengevoegd met het programma dat al in het geheugen aanwezig is. Na afsluiting van deze MERGE-bewerking blijft het samengevoegde programma in het werkgeheugen en keert GW-BASIC terug naar de commando-modus.

Als het gezochte programma niet als ASCII-bestand werd opgeslagen (met behulp van de a optie bij het commando SAVE), dan treedt de fout "Onjuiste bestandsmodus" op en wordt niets aan het programma in het werkgeheugen gewijzigd.

Als het bestaande programma en het daarmee samengevoegde diskettebestand programmaregels bevatten met gelijke regelnummers, dan worden de regels in het bestaande programma vervangen door de regels die in het diskette-bestand het gelijke regelnummer hebben.

Voorbeeld:

```
MERGE "SUBRTN"
```

Met behulp van dit commando wordt het bestand "SUBRTN.BAS" samengevoegd met het programma dat al in het werkgeheugen aanwezig is, vooropgesteld dat "SUBRTN" als ASCII-bestand op diskette werd opgeslagen. Als in de twee programma's regels met hetzelfde regelnummer voorkomen, worden regels uit het huidige programma vervangen door de regels uit het diskette-bestand die hetzelfde regelnummer hebben.

MID$ functie

Functie:

Een tekenreeks vormen, bestaande uit het opgegeven aantal tekens vanaf een opgegeven positie in een andere tekenreeks.

Syntaxis:

MID$(x$,*n*[,m])

Opmerkingen:

De waarde van *n* bepaalt vanaf welke positie in de bestaande tekenreeks de tekens voor de nieuwe tekenreeks moeten worden opgehaald. Voor *n* mag minimaal 1 en maximaal 255 worden opgegeven.

Argument *m* is optioneel en bepaalt hoeveel tekens uit de bestaande tekenreeks worden opgehaald voor de nieuwe tekenreeks. Voor *m* mag minimaal 0 en maximaal 255 worden opgegeven.

Als u voor *m* geen waarde opgeeft, of als de waarde van *m* groter is dan het aantal tekens rechts van *n* in de bestaande tekenreeks, dan bestaat de nieuwe tekenreeks uit alle tekens vanaf positie *n* in de bestaande tekenreeks.

Als *n* groter is dan de lengte van x$, dus n>LEN(x$), dan leidt MID$ tot een nul-string. Ook als *m* gelijk is aan 0 geeft MID$ een nul-string.

Als u voor *n* en/of *m* een waarde opgeeft die buiten het geldige bereik valt, dan volgt de foutmelding "Onjuiste functieaanroep".

Raadpleeg ook LEFT$ en RIGHT$ voor meer bijzonderheden.

Voorbeeld:

```
10 A$="Goede "
20 B$="morgen avond middag"
30 PRINT A$;MID$(B$,14,6)
RUN
Goede middag
Ok
```

Regel 30 geeft eerst A$ weer. Vervolgens worden uit B$ 6 tekens opgehaald, te beginnen op positie 14. Deze worden met A$ samengevoegd.

MID$ opdracht

Functie:

Vervangen van een deel van een tekenreeks door een andere tekenreeks.

Syntaxis:

MID$(*tekenreeks1*,*n*[,*m*])=*tekenreeks2*

Opmerkingen:

Zowel voor *n* als voor *m* moeten gehele getallen worden opgegeven.

De tekens in *tekenreeks1* worden, te beginnen vanaf positie *n*, vervangen door de tekens uit *tekenreeks2*.

Het optionele argument *m* bepaalt hoeveel tekens uit *tekenreeks2* worden gebruikt bij de vervangingsprocedure. Als u voor *m* geen waarde opgeeft, wordt *tekenreeks2* in zijn geheel gebruikt. Bij het vervangen wordt *tekenreeks1* nooit langer dan hij was voordat de vervanging wordt uitgevoerd, los van het feit of u een waarde voor *m* opgeeft en hoe groot die waarde is.

Voorbeeld:

```
10 A$="Heuvelstraat 24"
20 MID$(A$,14)="36"
30 PRINT A$
RUN
Heuvelstraat 36
Ok
```

Met behulp van de MID$-opdracht op regel 20 wordt het huisnummer 24 vervangen door 36.

MKDIR commando

Functie:
Aanmaken van een subdirectory.

Syntaxis:
MKDIR *padnaam*

Opmerkingen:
Voor *padnaam* geeft u een tekenreeks op van maximaal 63 tekens, die de naam voor de nieuwe subdirectory bevat.

Voorbeeld:

```
MKDIR "C:VERKOPEN\PETERS"
```

Dit commando maakt voor de verkoper Peters de subdirectory *peters* aan binnen de directory *verkopen*, opgeslagen in station C:.

MKI$, MKS$, MKD$ functies

Functie:
Numerieke waarden omzetten in tekenreeksen.

Syntaxis:
MKI$ (*geheel getal*)
MKS$ (*getal met enkele precisie*)
MKD$ (*getal met dubbele precisie*)

Opmerkingen:
MKI$ zet een geheel getal (integer) om in een tekenreeks van 2 bytes.

MKS$ zet een getal met enkele precisie om in een tekenreeks van 4 bytes.

MKD$ zet een getal met dubbele precisie om in een tekenreeks van 8 bytes.

Elke numerieke waarde die via LSET- of RSET-opdrachten in een bestandsbuffer wordt geplaatst, moet worden omgezet in een tekenreeks. Zie ook de beschrijving van CVI, CVS en CVD, die de omgekeerde bewerking uitvoeren.

De hier beschreven functies verschillen van STR$ omdat niet de bytes zelf worden veranderd (zoals bij STR$) maar alleen de wijze waarop de bytes worden geïnterpreteerd.

Voorbeeld:

```
100 BEDRAG=(K+T)
110 FIELD #1, 4 AS D$, 20 AS N$
120 LSET D$=MKS$(BEDRAG)
130 LSET N$=A$
140 PUT #1
```

NAME commando

Functie:

De naam van een bestand op schijf wijzigen.

Syntaxis:

NAME *oude bestandsnaam* AS *nieuwe bestandsnaam*

Opmerkingen:

De *oude bestandsnaam* moet bestaan en de *nieuwe bestandsnaam* mag nog niet bestaan, anders treedt er een fout op.

Na een NAME-commando is het bestand onder de nieuwe naam opgeslagen op dezelfde plaats op dezelfde diskette.

Voorbeeld:

```
NAME "KAS86" AS "KAS87"
Ok
```

Het bestand dat tot nu toe de bestandsnaam *kas86* had wordt met de nieuwe bestandsnaam *kas87* opgeslagen op dezelfde plaats op de diskette.

NEW commando

Functie:

Programma wissen dat zich in het werkgeheugen bevindt en aan alle variabelen de waarde 0 toekennen.

Syntaxis:

NEW

Opmerkingen:

Het commando NEW wordt gebruikt om het geheugen leeg te maken alvorens daarin een nieuw programma op te slaan. GW-BASIC keert na uitvoering van het NEW-commando altijd terug naar de commando-modus.

Voorbeeld 1:

```
NEW
Ok
```

Voorbeeld 2:

```
980 PRINT "Wilt u stoppen? (j/n)
990 ANTW$=INKEY$: IF ANTW$="" THEN 990
1000 IF ANTW$="j" THEN NEW
1010 IF ANTW$="n" THEN 300
1020 GOTO 990
```

OCT$ functie

Functie:
· Decimale getallen omzetten in octale getallen.

Syntaxis:
OCT$(*x*)

Opmerkingen:
Deze functie zet decimale getallen (uit het bereik -32768 tot +65535) om in getallen uit het octale stelsel.

Ons gangbare decimale stelsel is gebaseerd op het grondtal 10. Bij het octale stelsel wordt het grondtal 8 gebruikt. De appendices C en D geven meer bijzonderheden over octale getallen.

Het decimale getal (*x*) wordt eerst afgerond tot een geheel getal en daarna omgezet in een octaal getal.

Zie ook de beschrijving van HEX$ voor omzetting naar hexadecimale getallen (grondtal 16).

Voorbeeld:

```
10 PRINT OCT$(19)
RUN
23
ok
19 decimaal is 23 octaal (2*8^1 + 3*8^0)
Ok
```

ON COM(n), ON KEY(n), ON PEN, ON PLAY(n), ON STRIG(n), en ON TIMER(n) opdrachten

Functie:
Programma naar een bepaalde programmaregel sturen als een bepaalde gebeurtenis wordt gedetecteerd, bijvoorbeeld het indrukken van functie- of cursortoetsen, informatie op een communicatiepoort, gebruik van lichtpen of joysticks.

Syntaxis:
ON *gebeurtenis* **GOSUB** *regelnummer*

Opmerkingen:
De instructie stuurt het programma naar de achter GOSUB opgegeven programmaregel als de achter ON gespecificeerde gebeurtenis optreedt. Als het regelnummer 0 is, heeft de opdracht geen effect.

Zodra het programma naar het regelnummer is gestuurd, kunnen onderstaande opdrachten de gebeurtenis afhandelen:

gebeurtenis **ON** Als ON wordt gebruikt en tevens een regelnummer groter dan 0 is opgegeven, controleert BASIC bij de uitvoering van elke opdracht of de gebeurtenis is opgetreden. Als dat zo is, gaat BASIC via GOSUB naar het regelnummer dat in de ON-opdracht is opgegeven.

gebeurtenis **OFF** Bij gebruik van OFF wordt de gebeurtenis niet gedetecteerd.

gebeurtenis **STOP** Bij STOP wordt niet onmiddellijk op de gebeurtenis gereageerd, maar als hij optreedt, wordt dat wel onthouden en bij de eerstvolgende ON afgehandeld.

Als voor een bepaalde gebeurtenis een detectie wordt opgesteld, stopt het programma automatisch bij het optreden van die gebeurtenis. Zo wordt verhinderd dat onmiddellijk opnieuw op diezelfde gebeurtenis wordt gereageerd.

Als in de afhandelingsroutine niet expliciet OFF wordt opgegeven, wordt door die routine automatisch ON geactiveerd.

Als een opdracht voor foutdetectie wordt uitgevoerd nadat een fout is opgetreden, dan worden automatisch alle andere detecties uitgeschakeld.

Detectie van gebeurtenissen kan alleen worden uitgevoerd tijdens de uitvoering van een BASIC programma.

Voor gebeurtenis kunnen de volgende waarden worden opgegeven:

COM(*n*)	*n* is het nummer van de COM-poort (1 of 2)
KEY(*n*)	*n* is het nummer van een functietoets (1-20). De nummers 1-10 verwijzen naar functietoetsen F1 t/m F10; 11-14 naar de cursortoetsen: 11 = cursor omhoog 12 = cursor naar links 13 = cursor naar rechts 14 = cursor omlaag en 15-20 zijn door de gebruiker gedefinieerde toetsen.
PEN	Omdat er slechts één pen kan worden gebruikt, is geen nummer noodzakelijk.
PLAY(*n*)	De waarde voor *n* moet een geheel getal uit het bereik 1-32 zijn, anders treedt de fout "Onjuiste functieaanroep" op.
STRIG(*n*)	Voor *n* kunnen de waarden 0, 2, 4, of 6 worden opgegeven (0=vuurknop A1; 4=vuurknop A2, 2=vuurknop B1, 6=vuurknop B2).
TIMER(*n*)	De waarde van *n* moet in het bereik 1-86400 liggen, anders treedt de fout "Onjuiste functieaanroep" op.

RETURN *regelnummer* Deze vorm van RETURN wordt vooral gebruikt om het programma vanuit een routine voor detectie van gebeurtenissen terug te sturen naar een bepaald regelnummer. Zo wordt verhinderd dat het programma op de gebruikelijke wijze terugkeert naar de eerste opdracht achter de GOSUB waarmee de routine werd aangeroepen. Wees zeer zorgvuldig met toepassing van de constructie RETURN regelnummer, want GOSUB-, WHILE- en FOR-opdrachten die bij het aanroepen van de subroutine niet volledig waren afgehandeld, worden onthouden. Als de gebeurtenis binnen een subroutine is opgetreden, treedt de fout "NEXT zonder FOR" op als u probeert terug te keren naar een regel binnen een programmalus buiten de subroutine.

AANVULLENDE INFORMATIE VOOR ELK TYPE DETECTIE:

COM:

Een routine die communicatie van gegevens detecteert, voert de RETURN pas uit na het lezen van een complete melding op de COM-poort.

Gebruik geen COM-detectie voor een melding die slechts één teken lang is. In dat geval kan bij een hoge baud rate overlap optreden tussen detecteren en lezen van afzonderlijke tekens, waardoor de COM-buffer overvol raakt.

KEY:

De toetsen 15-20 zijn als volgt door de gebruiker te definiëren:

KEY(*n*),CHR$[*hexcode*]+CHR$[*scancode*]

Argument *n* moet een geheel getal zijn uit het bereik 15 t/m 20 en geeft aan welke toets moet worden gedetecteerd.

De *hexcode* is het masker voor de gedefinieerde toets: CAPS LOCK, NUM LOCK, ALT, CTRL, SHIFT-LINKS en SHIFT-RECHTS.

De *scancode* verwijst naar een van de 83 te programmeren toetsen. Appendix H geeft een overzicht van de scancodes voor deze toetsen.

Het juiste bit in de hexcode moet worden gezet om te detecteren dat behalve de toets ook SHIFT, CTRL of ALT is ingedrukt. De hexcodes luiden als volgt:

Masker	Hexcode	Duidt op
EXTENDED	&H80	Extended toets
CAPS LOCK	&H40	CAPS LOCK geactiveerd
NUM LOCK	&H20	NUM LOCK geactiveerd
ALT	&H08	ALT ingedrukt
CTRL	&H04	CTRL ingedrukt
SHIFT LINKS	&H02	Linker SHIFT-toets ingedrukt
SHIFT RECHTS	&H01	Rechter SHIFT-toets ingedrukt

Voor het detecteren van toetsen die samen met SHIFT worden ingedrukt, kunt u de waarden &H01, &H02 of &H03 gebruiken. Door alleen de waarde &H03 op te geven worden beide SHIFT-toetsen gedetecteerd.
Raadpleeg de beschrijving van de opdracht KEY(*n*) voor meer bijzonderheden.

Als GW-BASIC in de commando-modus wordt gebruikt, worden alle detectiefuncties uitgeschakeld. De functietoetsen voeren dan hun standaard voorgeprogrammeerde functie uit.

Als een toets is opgenomen in detectie-instructies, dan is die toets niet bruikbaar in opdrachten INPUT of INKEY$. Dit houdt in dat de detectieroutine voor elke toets anders moet zijn als een andere functie wordt vereist.

Als de toetscombinatie CTRL-PRTSC wordt gedetecteerd, dan wordt eerst de terugmelding van de printer verwerkt. Bij detectie van de toetscombinatie CTRL-PRTSC gaat het terugmelden van ontvangen tekens gewoon door als CTRL-PRTSC wordt ingedrukt.

De functietoetsen 1 t/m 14 zijn voorgeprogrammeerd. Daarom heeft het gebruik van de codes 59-68, 72, 75, 77 of 80 geen effect.

PLAY(n):

Detectie van PLAY is alleen mogelijk als achtergrondmuziek wordt gespeeld (PLAY"MB). Als deze detectie wordt uitgevoerd bij PLAY"MF (de standaard situatie) dan heeft deze detectie geen effect.

Kies de waarde voor *n* zeer zorgvuldig. Een opdracht als ON PLAY(32).. zal zo vaak een detectie activeren dat er nauwelijks tijd overblijft om het resterende programma uit te voeren.

De opdracht ON PLAY(*n*) zal een detectie uitvoeren zodra de wachtrij voor achtergrondmuziek overgaat van *n* naar *n-1* noten.

STRIG(n):

Via STRIG(*n*) ON activeert u de interrupt-routine die de status van de vuurknop controleert. Gedetecteerde bediening van de vuurknop heeft geen invloed op de instelling van de functies STRIG(0), STRIG(2), STRIG(4) of STRIG(6).

TIMER(n):

De opdracht ON TIMER(*n*) wordt vooral gebruikt in toepassingen waarvoor een interne timer vereist is, zoals het meten van reactietijd. Na de uitvoering van de opdracht ON TIMER wordt *n* seconden de gelegenheid geboden een handeling te verrichten.

Voorbeeld 1:

```
10 REM "ON COM(n)" voorbeeld
20 OPEN "COM1:9600,O,7" AS #1
30 ON COM(1) GOSUB 80
40 COM(1) ON
50 REM * Lees ingetikte tekens in *
60 A$=INKEY$: IF A$="" THEN 50
70 PRINT #1,A$;: GOTO 50
80 REM * Tekens naar beeldscherm *
90 ALL=LOC(1): IF ALL<1 THEN RETURN
100 B$=INPUT$(ALL,#1): PRINT B$;: RETURN
```

Voorbeeld 1 is een eenvoudig programma voor het detecteren van ingetikte gegevens.

Voorbeeld 2:

```
10 KEY 15,CHR$(4)+CHR$(70) 'Detect. CTRL BREAK
20 KEY 16,CHR$(12)+CHR$(83) 'Detect. reset systeem
30 ON KEY(15) GOSUB 1000
40 ON KEY(16) GOSUB 2000
50 KEY(15) ON
60 KEY(16) ON
.
.
.
1000 PRINT "Sorry, dit is niet toegestaan"
1010 RETURN
1020 POGING=POGING+1
2010 ON POGING GOTO 2100,2200,2300,2400,2500
2100 PRINT "Regel 2100, dus POGING was 1":RETURN
2200 PRINT "Tweede regelnummer, dus POGING was
     2":RETURN
2300 PRINT "POGING=3; naar regel 2300
     gesprongen":RETURN
2400 PRINT "Regel 2400":RETURN
2500 KEY(16) OFF 'Als de toets nogmaals wordt
2510 RETURN  'ingedrukt geeft BASIC de geest
```

Dit programma verhindert de mogelijkheid tot CTRL-BREAK of reset van het systeem tijdens de uitvoering van een programma.

Voorbeeld 3:

```
10 ON TIMER(60) GOSUB 1000
20 TIMER ON
.
.
.
1000 OUDRIJ=CSRLIN: REM Slaat huidige rij op
1010 OUDKOL=POS(0): REM Slaat huidige kolom op
1020 LOCATE 1,1: PRINT TIME$
1030 LOCATE OUDRIJ,OUDKOL: REM Slaat rij en kol
     weer op
1040 RETURN
```

Met dit programma wordt een 1 maal per minuut de tijd op de eerste schermregel afgebeeld.

ON COM(n) opdracht

Functie:

Hiermee wordt de eerste regel opgegeven van de subroutine die moet worden uitgevoerd wanneer er zich aktiviteiten aan een communicatiepoort voordoen.

Syntaxis:

ON COM(*n*) GOSUB *regelnummer*

Opmerkingen:

Het argument *n* is het nummer van de communicatiepoort.

Het *regelnummer* is het nummer van de eerste regel van de subroutine die moet worden uitgevoerd.

Het opgeven van regelnummer 0 schakelt de conditionele onderbreking voor communicatie uit.

De opdracht ON COM wordt alleen uitgevoerd als met behulp van de opdracht COM(*n*) ON conditionele onderbreking voor communicatie is ingeschakeld. Als dat zo is, en als het opgegeven regelnummer geen 0 is, controleert GW-BASIC na elke uitgevoerde opdracht of er aan de opgegeven communicatiepoort activiteit plaatsvindt. In dat geval wordt de GOSUB-opdracht uitgevoerd.

Als er een COM OFF-opdracht is uitgevoerd voor de betreffende communicatiepoort, wordt de GOSUB-opdracht niet uitgevoerd en wordt er niets onthouden.

Als er een COM STOP-opdracht is uitgevoerd voor de betreffende communicatiepoort, wordt de GOSUB-opdracht niet meteen uitgevoerd, maar wordt onthouden en zal direct worden uitgevoerd na de eerstvolgende COM ON-opdracht.

Tijdens de uitvoering van de GOSUB-opdracht vanwege een conditionele onderbreking, wordt automatisch een COM STOP van kracht, zodat recursieve condities uitgesloten zijn. Bij het einde van de subroutine (na de RETURN-opdracht), wordt COM ON automatisch uitgevoerd, tenzij binnen de subroutine COM OFF voorkwam.

Aan de RETURN-opdracht bij het eind van de subroutine mag een regelnummer worden toegevoegd. Gebruik dit echter voorzichtig, want eventuele andere GOSUB-, WHILE-, of FOR-opdrachten die al actief waren op het moment dat de GOSUB-opdracht wordt uitgevoerd, blijven actief. Daardoor kunnen "FOR zonder NEXT" fouten ontstaan.

Conditionele onderbreking vindt niet plaats als GW-BASIC geen programa aan het uitvoeren is, of als een programma onderbroken werd in verband met een gevonden fout.

ON ERROR GOTO opdracht

Functie:

Detecteren van opgetreden fouten en vervolgens naar een opgegeven regelnummer gaan.

Syntaxis:

ON ERROR GOTO *regelnummer*

Opmerkingen:

Zodra via deze opdracht het detecteren van fouten is geactiveerd, zal het programma na een fout (ook in de commando-modus, zoals syntaxis-fouten) naar het opgegeven regelnummer springen, waar een routine begint voor afhandeling van de opgetreden fout.

GW-BASIC zal de instructies in de routine uitvoeren en pas terugspringen naar het hoofdprogramma als in de routine een RESUME-instructie wordt uitgevoerd.

Als u een *regelnummer* opgeeft dat niet bestaat dan verschijnt de foutmelding "Ongedefinieerde regel".

Als u het detecteren van fouten wilt uitschakelen, gebruik dan de opdracht ON ERROR GOTO 0. Als daarna een fout zou optreden, verschijnt de bijbehorende foutmelding en stopt de uitvoering van het programma. Neem in alle routines voor foutafhandeling de opdracht ON ERROR GOTO 0 op voor het geval dat een fout wordt gedetecteerd die niet binnen de routine wordt afgehandeld.

Als binnen een routine voor foutafhandeling ON ERROR GOTO 0 wordt opgenomen, verschijnt de foutmelding en breekt GW-BASIC de uitvoering van het programma af. Binnen een routine voor foutafhandeling zelf worden fouten niet gedetecteerd.

Voorbeeld:

```
10 ON ERROR GOTO 1000
.
.
.
1000 A=ERR:B=ERL
1010 PRINT A,B
1020 RESUME NEXT
```

Regel 1010 geeft op het scherm het type fout en de plaats van de fout aan (zie de variabelen ERR en ERL).

Regel 1020 zorgt dat de programma-uitvoering wordt hervat op de regel na de regel waar de fout is opgetreden.

ON...GOTO en ON...GOSUB opdrachten

Functie:

Naar één van de opgegeven regelnummers springen, gekozen op basis van de waarde van een expressie in de opdracht.

Syntaxis:

ON *expressie* **GOTO** *regelnummers*
ON *expressie* **GOSUB** *regelnummers*

Opmerkingen:

In de opdracht ON...GOTO bepaalt de waarde van de *expressie* naar welk *regelnummer* uit de lijst regelnummers wordt gesprongen. Als de uitkomst van de expressie bijvoorbeeld 3 is, springt het programma naar het derde regelnummer dat achter GOTO of GOSUB is opgegeven. De uitkomst van de expressie wordt indien nodig eerst afgerond tot een geheel getal.

In de opdracht ON...GOSUB moet elk van de *regelnummers* verwijzen naar de eerste regel van een subroutine.

Als de waarde van de *expressie* 0 is of een getal dat groter is dan het aantal regelnummers achter GOTO of GOSUB (maximaal 255), dan springt GW-BASIC naar de eerstvolgende uitvoerbare opdracht. Als de uitkomst van de expressie negatief is of groter is dan 255 dan treedt de fout "Onjuiste functieaanroep" op.

Voorbeeld 1:

```
100 IF R<1 OR R>4 THEN PRINT "Error": END
```

Als de gehele waarde van R kleiner dan 1 of groter dan 4 is, stopt de uitvoering van het programma.

Voorbeeld 2:

```
200 ON R GOTO 300,400,500,600
```

Als R gelijk is aan 1, dan springt GW-BASIC naar regel 300, als R gelijk is aan 2, naar regel 400, enz.

ON KEY(n) opdracht

Functie:

Verwijzen naar de eerste regel van een subroutine, die moet worden uitgevoerd als de opgegeven toets is ingedrukt.

Syntaxis:

ON KEY(*n*) GOSUB *regelnummer*

Opmerkingen:

Het argument *n* is het nummer van een functietoets, cursortoets of door de gebruiker zelf geprogrammeerde toets.

Het *regelnummer* moet overeenkomen met de eerste regel van een subroutine. Deze wordt uitgevoerd zodra de opgegeven toets is ingedrukt. Als het *regelnummer* 0 is, wordt de detectie uitgeschakeld.

De opdracht ON KEY wordt alleen maar uitgevoerd als vooraf een KEY(*n*) ON-opdracht werd uitgevoerd om het detecteren van gebeurtenissen te activeren. Als dat zo is en bovendien een *regelnummer* ongelijk aan 0 is opgegeven, dan controleert GW-BASIC voorafgaand aan de uitvoering van elke instructie of u de opgegeven toets hebt ingedrukt. Is dat zo, dan springt het programma naar de subroutine die door het opgegeven regelnummer wordt aangeduid.

Als de toets niet langer wordt gedetecteerd omdat een KEY(*n*) OFF-opdracht werd uitgevoerd, dan wordt niet gereageerd op het indrukken van de toets. Als een KEY(*n*) STOP-opdracht is uitgevoerd wordt niet onmiddellijk gereageerd op het indrukken van de opgegeven toets. Het systeem heeft echter wel onthouden dat die toets werd ingedrukt. Bij de eerstvolgende KEY(*n*) ON-opdracht springt het programma dan alsnog naar de subroutine.

De reactie op de detectie van een gebeurtenis (dus uitvoering van de GOSUB) leidt automatisch tot uitvoering van KEY(*n*) STOP, zodat de gebeurtenis niet onmiddellijk opnieuw wordt gedetecteerd. RETURN aan het einde van de subroutine voert automatisch KEY(*n*) ON uit als in de subroutine zelf geen instructie KEY(*n*) OFF staat.

Via RETURN met *regelnummer* kunt u vanuit de subroutine terugspringen naar een bepaalde regel. Gebruik dit echter zeer zorgvuldig, want het systeem onthoudt GOSUB-, WHILE- en FOR-opdrachten die bij het aanroepen van de subroutine niet volledig waren afgehandeld, zodat fouten als "NEXT zonder FOR" kunnen optreden als u naar een onjuist regelnummer springt.

Detectie van gebeurtenissen functioneert alleen tijdens de uitvoering van een BASIC programma. Als een fout optreedt en wordt afgehandeld, is de detectie van gebeurtenissen uitgeschakeld.

Voor detectie van ingedrukte toetsen geldt:

1 Eerst wordt een print-opdracht verwerkt. Als de toetsencombinatie CTRL-PRTSC wordt gedetecteerd, gaat het terugmelden van ontvangen tekens gewoon door.

2 Vervolgens controleert GW-BASIC of een functietoets of cursortoets werd ingedrukt. Expliciet via een instructie detecteren van deze toetsen heeft geen zin, want ze zijn al voorgeprogrammeerd.

3 Tenslotte volgt detectie van de toetsen die door de gebruiker werden opgegeven.

4 Als het indrukken van een toets wordt gedetecteerd, onthoudt BASIC niet welke toets dat was.

LET OP:
Detectie van het indrukken van een toets werkt voor alle toetsen, zelfs voor BREAK en reset (warm opstarten, d.w.z. zonder de computer uit te schakelen). Dat betekent dus dat u via het detecteren van toetsen zelfs kunt verhinderen dat u per abuis een programma verlaat of een reset op het systeem uitvoert. Als het indrukken van een gedefinieerde toets wordt gesignaleerd, onthoudt het systeem niet welke toets dat was. U kunt dat dus niet nagaan via INPUT of INKEY$. Als u bepaalde functies onder enkele toetsen wilt opslaan, moet u voor elk van die toetsen afzonderlijke subroutines gebruiken. Programmeer dus nooit meerdere functies binnen één subroutine.

Via de opdracht ON KEY(*n*) kunnen zes willekeurige extra toetsen worden gedetecteerd (ook CONTROL, SHIFT of ALT), en wel als volgt:

```
ON KEY(<n>) GOSUB <regelnummer>
```

waarbij *n* een geheel getal is voor een geldig toetsnummer.

Voorbeeld 1:

```
10 KEY 4,"SCREEN 0,0" 'Functietoets 1
   geprogrammeerd
20 KEY(4) ON           'Detectie geactiveerd
.
.
70 ON KEY(4) GOSUB 200
.
.   (functietoets 4 wordt ingedrukt)
.
200 'Subroutine voor beeldscherm
```

De functie van functietoets 4 werd vervangen door de instructie SCREEN 0,0, die wordt uitgevoerd zodra de toets wordt ingedrukt. De toets krijgt zijn oorspronkelijke functie terug als het systeem opnieuw wordt opgestart.

Voorbeeld 2:

```
100 KEY 15, CHR$(&H04) + CHR$(83)
105 REM Toets 15 is nu gelijk aan Control-S
110 KEY(15) ON
.
.
1000 PRINT "Als u het programma wilt laten pauzeren,"
1010 PRINT "druk dan op de toetscombinatie Control-S."
1020 ON KEY(15) GOSUB 3000
.
.   (combinatie Control-S wordt ingedrukt)
.
3000 REM ** Onderbreek programma
3010 CLOSE 1
3020 RESET
3030 CLS
3035 PRINT "Tik CONT in voor voortzetting."
3040 STOP
3050 OPEN "A",1, "DEBITEUR.DAT"
3060 RETURN
```

In dit voorbeeld roept de geprogrammeerde toetsencombinatie CONTROL-S een subroutine aan die bestanden sluit, het programma stopt en dan wacht tot de instructie CONT wordt ingetikt.

ON PEN opdracht

Functie:

Verwijzen naar de subroutine die moet worden uitgevoerd zodra de lichtpen wordt gebruikt.

Syntaxis:

ON PEN GOSUB *regelnummer*

Opmerkingen:

Het *regelnummer* verwijst naar de eerste regel van de subroutine die moet worden aangeroepen zodra de lichtpen wordt gedetecteerd. Als *regelnummer* 0 wordt opgegeven, wordt de detectie uitgeschakeld.

ON PEN functioneert alleen nadat de detectie is geactiveerd via de opdracht PEN ON en bovendien het regelnummer bij ON PEN ongelijk is aan 0. GW-BASIC controleert steeds of de lichtpen wordt gebruikt en zodra dat wordt gedetecteerd, springt het programma naar de opgegeven subroutine.

Nadat een instructie PEN OFF is uitgevoerd, wordt de detectie uitgeschakeld en wordt GOSUB dus niet uitgevoerd.

Als een instructie PEN STOP is uitgevoerd, wordt de detectie onthouden en onmiddellijk na de eerstvolgende PEN ON-opdracht wordt de GOSUB-instructie alsnog uitgevoerd.

Na het uitvoeren van de instructie GOSUB wordt automatisch PEN STOP uitgevoerd, zodat dezelfde gebeurtenis niet onmiddellijk opnieuw wordt gedetecteerd. RETURN aan het einde van de subroutine voert automatisch PEN ON uit als in de subroutine zelf geen instructie PEN OFF staat.

Via RETURN met *regelnummer* kunt u vanuit de subroutine terugspringen naar een bepaalde regel. Gebruik dit echter zeer zorgvuldig, want het systeem onthoudt GOSUB-, WHILE- en FOR-opdrachten die bij het aanroepen van de subroutine niet volledig waren afgehandeld, zodat een fout als "NEXT zonder FOR" kan optreden als u naar een onjuist regelnummer springt.

Detectie van gebeurtenissen functioneert alleen tijdens de uitvoering van een BASIC programma. Tijdens het uitvoeren van een routine voor foutafhandeling is de detectie van gebeurtenissen uitgeschakeld.

ON PLAY opdracht

Functie:

Naar een subroutine springen als de muziek-wachtrij minder dan een opgegeven *aantal noten* bevat. Dit maakt het mogelijk tijdens het programma continu muziek weer te geven.

Syntaxis:

ON PLAY (*aantal noten*) **GOSUB** *regelnummer*

Opmerkingen:

Het *aantal noten* moet een geheel getal zijn tussen 1 en 32. Andere waarden leiden tot de fout "Onjuiste functieaanroep".

Het argument *regelnummer* verwijst naar de eerste regel van de subroutine voor detectie van PLAY. De opdracht ON PLAY detecteert wanneer de lengte van de wachtrij overgaat van (*aantal noten*) naar (*aantal noten* - 1).

ON PLAY wordt gebruikt in combinatie met PLAY ON, PLAY OFF en PLAY STOP, die respectievelijk de detectie inschakelen, uitschakelen en onderbreken.

Na uitvoering van PLAY ON wordt de GOSUB-instructie uitgevoerd als PLAY wordt gedetecteerd.

Na uitvoering van PLAY OFF wordt niets gedetecteerd en GOSUB niet uitgevoerd.

Na uitvoering van PLAY STOP wordt GOSUB niet uitgevoerd. De detectie wordt echter onthouden en na de eerstvolgende opdracht PLAY ON wordt alsnog gereageerd door de GOSUB-instructie uit te voeren.

Na het uitvoeren van de instructie GOSUB wordt automatisch PLAY STOP uitgevoerd, zodat dezelfde gebeurtenis niet onmiddellijk opnieuw wordt gedetecteerd. RETURN aan het einde van de subroutine voert automatisch PLAY ON uit als in de subroutine zelf geen instructie PLAY OFF staat.

Via RETURN met *regelnummer* kunt u vanuit de subroutine terugspringen naar een bepaalde regel. Gebruik dit echter zeer zorgvuldig, want het systeem onthoudt GOSUB-, WHILE- en FOR-opdrachten die bij het aanroepen van de subroutine niet volledig waren afgehandeld, zodat fouten als "NEXT zonder FOR" kunnen optreden als u naar een onjuist regelnummer springt.

Bij deze opdracht gelden de volgende regels:

1 Detectie van PLAY functioneert alleen tijdens het afspelen van achtergrondmuziek, nooit tijdens voorgrondmuziek.

2 Als de lengte van de wachtrij al is overgegaan van n naar n-1 noten voordat PLAY ON wordt uitgevoerd, dan werkt de detectie niet.

3 Als voor *n* een groot getal wordt opgegeven, vindt de detectie zo vaak plaats dat de programma-uitvoering wordt vertraagd.

Zie ook de beschrijving van PLAY ON, PLAY OFF, PLAY STOP.

Voorbeeld:

```
100 PLAY ON
.
.
540 PLAY "MB L1 XPIANO$"
550 ON PLAY(8) GOSUB 1000
.
.
.
1000 REM **Achtergrondmuziek **
1010 LET MAAT% = MAAT% + 1
.
.
1099 RETURN
```

Het programma roept de subroutine aan zodra de lengte van de wachtrij daalt tot 7 noten.

ON STRIG(n) opdracht

Functie:

Naar een subroutine springen zodra de vuurknop van een joystick wordt ingedrukt.

Syntaxis:

ON STRIG(*n*) GOSUB *regelnummer*

Opmerkingen:

Argument *n* verwijst naar het nummer van de vuurknop.

Het *regelnummer* verwijst naar de eerste regel van de subroutine die moet worden uitgevoerd zodra de vuurknop wordt ingedrukt. Als het *regelnummer* 0 wordt opgegeven, wordt het indrukken van de vuurknop niet langer gedetecteerd.

De subroutine wordt alleen aangeroepen indien vooraf een STRIG ON-opdracht werd uitgevoerd om de detectie te activeren en als het *regelnummer* in ON STRIG ongelijk is aan 0. Als aan die voorwaarden is voldaan, controleert GW-BASIC steeds of de vuurknop werd ingedrukt. Als dat zo is wordt de subroutine aangeroepen.

Na uitvoering van STRIG OFF wordt niet gedetecteerd en wordt GOSUB dus niet uitgevoerd.

Na uitvoering van STRIG STOP wordt de detectie onthouden en wordt na de eerstvolgende opdracht STRIG ON alsnog de GOSUB-instructie uitgevoerd.

Na het uitvoeren van de instructie GOSUB wordt automatisch STRIG STOP uitgevoerd, zodat dezelfde gebeurtenis niet onmiddellijk opnieuw wordt gedetecteerd. RETURN aan het einde van de subroutine voert automatisch STRIG ON uit als in de subroutine zelf geen instructie STRIG OFF staat.

Via RETURN met regelnummer kunt u vanuit de subroutine terugspringen naar een bepaalde regel. Gebruik dit echter zeer zorgvuldig, want het systeem onthoudt GOSUB-, WHILE- en FOR-opdrachten die bij het aanroepen van de subroutine niet volledig waren afgehandeld, zodat fouten als "NEXT zonder FOR" kunnen optreden als u naar een onjuist regelnummer springt.

De detectie van de vuurknop functioneert alleen tijdens het uitvoeren van een GW-BASIC programma en functioneert niet als een fout werd gedetecteerd en tijdens de afhandeling daarvan.

ON TIMER(n) opdracht

Functie:
Steeds na een bepaald tijdsinterval een subroutine aanroepen.

Syntaxis:
ON TIMER(*n*) **GOSUB** *regelnummer*

Opmerkingen:
Via ON TIMER wordt steeds na een interval van *n* seconden een subroutine aangeroepen. De waarde van *n* moet tussen 1 en 86400 liggen (1 seconde tot 24 uur). Waarden buiten dit bereik leiden tot de fout "Onjuiste functieaanroep".

De subroutine wordt alleen aangeroepen indien vooraf een TIMER ON-opdracht werd uitgevoerd om de detectie te activeren en als het regelnummer in ON TIMER ongelijk is aan 0. Als aan die voorwaarden is voldaan, controleert GW-BASIC steeds of het opgegeven aantal seconden is verstreken. Zodra dat zo is, wordt de subroutine aangeroepen.

Na uitvoering van TIMER OFF wordt het verstrijken van de opgegeven tijd niet gedetecteerd en wordt GOSUB dus niet uitgevoerd.

Na uitvoering van TIMER STOP wordt wel gedetecteerd dat de tijd is verstreken, maar daarop wordt pas gereageerd na de eerstvolgende opdracht TIMER ON.

Na het uitvoeren van de instructie GOSUB wordt automatisch TIMER STOP uitgevoerd, zodat dezelfde subroutine niet onmiddellijk opnieuw wordt aangeroepen. RETURN aan het einde van de subroutine voert automatisch TIMER ON uit als in de subroutine zelf geen instructie TIMER OFF staat.

Via RETURN met *regelnummer* kunt u vanuit de subroutine terugspringen naar een bepaalde regel. Gebruik dit echter zeer zorgvuldig, want het systeem onthoudt GOSUB-, WHILE- en FOR-opdrachten die bij het aanroepen van de subroutine niet volledig waren afgehandeld, zodat een fout als "NEXT zonder FOR" kan optreden als u naar een onjuist *regelnummer* springt.

Voorbeeld:

```
20 TIMER ON
.
.
.
1000 OUDRIJ=CSRLIN 'Sla huidige rij op
1010 OUDKOL=POS(0) 'Sla huidige kolom op
1020 LOCATE 1,1: PRINT TIME$
1030 LOCATE OUDRIJ,OUDKOL 'Sla rij en kolom weer op
1040 RETURN
```

Met dit programma wordt na elke minuut op regel 1 de tijd afgebeeld.

Zie ook de beschrijving van TIMER ON, TIMER OFF en TIMER STOP.

OPEN opdracht

Functie:

Een bestand of randapparaat toegankelijk maken voor invoer/uitvoer (I/O).

Syntaxis:

OPEN *modus*,[#]*bestandsnummer*,*bestandsnaam*[,*rec-lengte*]

OPEN *bestandsnaam* [FOR *modus*][ACCESS *acc*][*lock*] AS[#] *bestandsnummer* [LEN=*rec-lengte*]

Opmerkingen:

Het argument *bestandsnaam* moet de naam van het bestand bevatten.

In de eerstgenoemde syntaxis is het argument *modus* een tekenreeks-expressie die bestaat uit een van de volgende letters:

Letter	Duidt aan
O	sequentiële uitvoer
I	sequentiële invoer
R	onbepaalde invoer/uitvoer
A	ga naar einde van het bestand

In de tweede syntaxis bepaalt het argument *modus* de startpositie in het bestand en ook welke handeling moet worden uitgevoerd als de opgegeven bestandsnaam niet bestaat. Als [FOR modus] niet wordt gebruikt, begint de in- of uitvoer aan het begin van het bestand. Als het opgegeven bestand niet wordt gevonden, dan wordt dat bestand aangemaakt. Dat is de onbepaalde-modus (I/O), waarbij records vanaf elke positie kunnen worden ingelezen of weggeschreven. Onderstaand overzicht geeft de modi aan en de uitgevoerde handelingen:

INPUT	Invoer vanaf het begin van het bestand. Als het opgegeven bestand niet bestaat, volgt de foutmelding "Bestand niet gevonden".
OUTPUT	Uitvoer vanaf het begin van het bestand. Als het opgegeven bestand niet bestaat, wordt deze aangemaakt.
APPEND	Vanaf het einde van het bestand. Als het opgegeven bestand niet bestaat, wordt deze aangemaakt.
RANDOM	Verwijst naar onbepaalde invoer/uitvoer-modus.

Het argument *modus* moet een tekenreeks-constante zijn, die niet tussen aanhalingstekens mag staan.

Voor het argument *acc* kan worden opgegeven:

READ
WRITE
READ WRITE

Voor *bestandsnummer* moet een waarde worden opgegeven die minimaal 1 is en maximaal het aantal toelaatbare bestanden. Het nummer koppelt een I/O-buffer aan een diskette-bestand of randapparaat. Die koppeling blijft bestaan tot een CLOSE-opdracht of een CLOSE-opdracht met bestandsnummer wordt uitgevoerd.

Voor *rec-lengte* moet een waarde tussen 1 en 32767 worden opgegeven, waarmee de record-lengte voor onbepaald benaderbare bestanden wordt bepaald. Als u zelf geen waarde opgeeft, hanteert de computer de standaard record-lengte van 128 bytes.

Bij sequentiële bestanden is de standaard record-lengte 128 bytes. Bovendien geldt dat *rec-lengte* niet groter kan zijn dan de waarde die op de commandoregel via /s werd geactiveerd.

Alvorens de computer gegevens kan uitwisselen met een schijfbestand, moet deze worden geopend. OPEN reserveert een buffer voor I/O-bewerkingen met dat bestand en bepaalt de toegangs-modus.

Als u verschillende bestandsnummers gebruikt, is het mogelijk tegelijkertijd meerdere bestanden te openen voor I/O. De volgende twee opdrachten zijn dus toegestaan:

```
OPEN "B:TEMP" FOR INPUT AS #1
OPEN "B:TEMP" FOR INPUT AS #2
```

U kunt een bestand echter slechts één maal openen voor APPEND of OUTPUT, zodat combinatie van de volgende twee opdrachten fout is:

```
OPEN "B:TEMP" FOR OUTPUT AS #1
OPEN "B:TEMP" FOR OUTPUT AS #2
```

Attentie:
Vergeet niet alle bestanden te sluiten alvorens de diskette uit het station te verwijderen. Zie CLOSE en RESET.

U kunt naar de volgende onderdelen van uw systeem verwijzen:

A:, B:, C: ...	station
KYBD:	toetsenbord (alleen voor invoer)
SCRN:	beeldscherm (alleen voor uitvoer)
LPT1:	printer 1
LPT2:	printer 2
LPT3:	printer 3
COM1:	RS-232 communicatiepoort 1
COM2:	RS-232 communicatiepoort 2

Voor elk van de onderdelen zijn de volgende OPEN-modi mogelijk:

KYBD: alleen Invoer

SCRN:	alleen Uitvoer
LPT1:	alleen Uitvoer
LPT2:	alleen Uitvoer
LPT3:	alleen Uitvoer
COM1:	Invoer, Uitvoer, of Random
COM2:	Invoer, Uitvoer, of Random

Voor diskette-bestanden zijn alle modi toegestaan.

Als een diskette-bestand wordt geopend voor APPEND, is de positie in eerste instantie het einde van het bestand en wordt het laatste record-nummer in het bestand gekozen (LOF(*x*)/128). PRINT-, WRITE- of PUT-instructies breiden het bestand vervolgens uit. Via een GET-opdracht wordt een andere positie in het bestand gekozen. Dit houdt dan tevens in dat de modus Random wordt gekozen.

Als u een andere positie dan het einde van het bestand kiest, worden nieuwe records aan het bestand toegevoegd door het uitvoeren van de opdracht GET #x,LOF(*x*)/*rec-lengte*. Deze instructie verplaatst het bestand pointer naar het einde van het bestand als voorbereiding voor de APPEND-bewerking.

Als voor een of meer argumenten een onjuiste waarde wordt opgegeven, verschijnt de foutmelding "Onjuiste functieaanroep" en wordt het bestand niet geopend.

Als een bestand wordt geopend voor INPUT zal bij pogingen gegevens naar dat bestand te schrijven de foutmelding "Onjuist bestandsnummer" verschijnen.

Als een bestand wordt geopend voor OUTPUT zal bij pogingen gegevens van dat bestand te lezen de foutmelding "Onjuist bestandsnummer" verschijnen.

Als een bestand al geopend is voor een willekeurige modus, dan zal een poging tot openen voor APPEND of OUTPUT mislukken.

Omdat in een subdirectory via verschillende paths (paden) naar hetzelfde bestand kan worden verwezen, kan GW-BASIC nauwelijks ontdekken of het inderdaad hetzelfde bestand betreft wanneer alleen het pad is opgegeven. Daarom kunt u een bestand op dezelfde diskette niet openen voor APPEND of OUTPUT, ook niet als het pad verschillend is. Stel dat u de directory *peters* als werk-directory gebruikt. In dat geval verwijzen alle onderstaande opdrachten naar hetzelfde bestand:

```
OPEN "RAPPORT"
OPEN "\VERKOPEN\PETERS\RAPPORT"
OPEN "..\PETERS\RAPPORT"
OPEN "..\..\PETERS\RAPPORT"
```

Het is toegestaan één bestandsnaam tegelijkertijd te openen onder meerdere bestandsnummers. Voor elk bestandsnummer wordt een andere buffer gereserveerd, zodat meerdere records uit hetzelfde bestand tegelijk in het geheugen beschikbaar zijn. Op die wijze kunt u verschillende modi voor verschillende toepassingen gebruiken. Tevens is het mogelijk voor afzonderlijke modi verschillende bestandsnummers te kiezen, waardoor u een programma overzichtelijker kunt maken.

Als de optie LEN=*rec-lengte* wordt gebruikt, mag *rec-lengte* niet groter zijn dan de record-lengte die op de commandoregel via de optie */s:rec-lengte* werd ingesteld.

Binnen een netwerk is gebruik van de opdracht OPEN beperkt tot twee verschillende combinaties van omstandigheden:

- Systeemonderdelen worden binnen het netwerk alleen voor bepaalde doeleinden gezamenlijk gebruikt, zodat OPEN-opdrachten slechts voor een deel van de opgegeven modi mogelijk zijn, zoals (I)nput, (O)utput, (A)ppend en de standaard modus (R)andom.
- Een bestand kan voor bepaalde bewerkingen worden afgesloten door een bepaalde vorm van de OPEN-opdracht. Dit garandeert dat het bestand voor een bepaalde modus exclusief gereserveerd blijft voor één gebruiker zolang de betreffende OPEN-opdracht geactiveerd is

Deze afsluiting (*lock*) kan betrekking hebben op:

SHARED	De modus "deny none" (geen beperkingen). Er is geen beperking op de toegankelijkheid van het bestand voor lezen/schrijven, maar de standaard modus wordt binnen geen van de modi (incl. SHARED) niet toegestaan.
LOCK READ	De modus "deny read" (lezen onmogelijk). Zodra een bestand is geopend voor LOCK READ wordt aan geen ander proces toegang verleend voor lezen. Als een gebruiker tracht de op deze wijze beperkte bestand te openen, slaagt hij daar niet in als het bestand al in de standaard modus geopend is of voor lezen.
LOCK WRITE	De modus "deny write" (schrijven onmogelijk). Als een bestand is geopend met de toevoeging LOCK WRITE, wordt aan geen ander proces toegang verleend voor schrijven. Het openen van dat bestand is niet mogelijk als het bestand in de standaard modus is geopend of door een ander programma is geopend met WRITE-beperking.
LOCK READ WRITE	De modus "deny all", of "exclusive". Als een WRITE-bestand met deze beperking door een programma is geopend, heeft alleen dat programma toegang tot het bestand. Geen enkel ander programma kan het bestand voor welke modus dan ook openen.
default	De standaard modus "compatibility", tijdens welke het bestand compatibel is met andere BASIC's en geen beperking geldt. Zolang het bestand niet is geopend, kan elk programma dit bestand openen, daarna heeft alleen het programma toegang waarmee het bestand werd geopend. Het bestand is dus voor één toepassing toegankelijk.

Als een bestand al is geopend door een bepaald programma, dan zal bij elke andere gebruiker de melding "Toegang geweigerd" verschijnen als hij probeert datzelfde bestand te openen. Dit gebeurt bijvoorbeeld als wordt geprobeerd een bestand via OPEN SHARED te benaderen terwijl dat bestand op dat moment al door een ander programma is geopend met OPEN LOCK READ WRITE.

Als een OPEN-opdracht niet tot succes leidt omdat de aangegeven modus incompatibel is met de in het netwerk gehanteerde beperkingen voor het betreffende bestand, dan treedt de fout "Pad/bestand niet toegankelijk" op. Dit zal bijvoorbeeld gebeuren als wordt geprobeerd een bestand te openen voor uitvoer in een directory die alleen gezamenlijk mag worden gebruikt voor lezen, maar niet voor schrijven.

Raadpleeg LOCK en UNLOCK voor het gebruik van bestanden in netwerken.

Voorbeeld:

```
10 OPEN "I",2,"KREDIET"
```

Dit programma opent bestand 2, "KREDIET", voor sequentiële invoer.

OPEN "COM(n) opdracht

Functie:

Een buffer openen voor asynchrone communicatie via de RS-232 poort met andere computers en randapparatuur, op dezelfde wijze als OPEN voor diskette-bestanden.

Syntaxis:

OPEN "COM[*n*]:[*snelhd*][,*par*][,*gegevens*] [,*stop*][,RS][,CS[*n*]] [,DS[*n*]] [,CD[*n*]][,LF] [,PE]" AS [#]*bestandsnummer* [LEN=*getal*]

Opmerkingen:

COM(*n*) is de aanduiding van de communicatiepoort: COM1: of COM2:.

Met het argument *snelhd* wordt de baud rate (transmissiesnelheid) opgegeven. Daarbij zijn de volgende snelheden toegestaan: 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 en 9600. De standaardwaarde is 300 baud (1 baud is gelijk aan 1 bit per seconde).

Het argument *par* bestaat uit één teken en duidt op de pariteit tijdens het verzenden en ontvangen van gegevens. De mogelijkheden voor de pariteit zijn:

S	SPACE. Het pariteitsbit wordt altijd verzonden en ontvangen als spatie (0 bit).
M	MARK. Het pariteitsbit wordt altijd verzonden en ontvangen als teken (1 bit).
O	ODD (Oneven). Oneven pariteit bij verzenden. Het programma gaat na of de som van de ontvangen reeks binaire tekens oneven is. Standaard is Even.
E	EVEN. Even pariteit bij verzenden. Het programma gaat na of de som van de ontvangen reeks binaire tekens even is. Dit is tevens de standaard pariteitscontrole.
N	NONE (Geen). Geen pariteit bij verzenden. Geen pariteitscontrole bij ontvangen.

Het argument *gegevens* moet bestaan uit een geheel getal, dat verwijst naar het aantal bits dat wordt ontvangen/verzonden. Het aantal bits mag worden ingesteld op 4, 5, 6, 7 of 8. Het standaard aantal is 7 bits.

Attentie:
Het is niet toegestaan 4 bits op te geven, gecombineerd met NONE voor de pariteit. Ook 8 bits met enigerlei vorm van pariteit is verboden.

Het argument *stop* bepaalt het aantal stop-bits. Dat mag 1 of 2 zijn. Als geen waarde wordt opgegeven, versturen de baud rates 75 en 110 bps twee stop-bits en alle andere transmissiesnelheden één stop-bit.

Voor *bestandsnummer* moet een waarde worden opgegeven van minimaal 1 en maximaal het aantal bestanden dat is toegestaan. Communicatie met een systeemonderdeel kan slechts voor één bestandsnummer tegelijkertijd plaatsvinden. Het *bestandsnummer* blijft naar het betreffende bestand verwijzen zolang dat bestand geopend blijft. Het wordt gebruikt om in andere COM I/O opdrachten naar het bestand te verwijzen.

Fouten in de bestandsnaam (expressie tussen aanhalingstekens) leiden tot de fout "Ongeldige bestandsnaam". Daarbij wordt niet aangegeven in welke argument(en) de fout zich bevindt.

Het argument *getal* duidt op het aantal bytes dat uit de communicatiebuffer kan worden gelezen wanneer de voor GET of PUT geldende standaardlengte van 128 bytes wordt gebruikt.

Als de computer geen signaal "Data Set Ready" (DSR) detecteert, of als dat tijdens het gegevenstransport wegvalt, treedt de fout "Eenheid niet beschikbaar" ("Maximale tijd verstreken") op.

De optionele argumenten RS, CS, DS, DC, LF en PE hebben de volgende functies:

Optie	Functie
RS	onderdrukt RTS (Request To Send)
CS[*n*]	bestuurt CTS (Clear To Send)
DS[*n*]	bestuurt DSR (Data Set Ready)
CD[*n*]	bestuurt CD (Carrier Detect)
LF	verzendt een line feed na elke return
PE	schakelt pariteitscontrole in

In deze argumenten geeft *n* (0-65535) in milliseconden aan hoe lang het systeem op het betreffende signaal moet wachten zonder de fout "Eenheid niet beschikbaar" te genereren. De standaardwaarden zijn CS1000, DS1000 en CD0. Als voor RS een waarde werd opgegeven, is de standaardwaarde voor CS gelijk aan 0. Als voor *n* geen waarde wordt opgegeven, wordt de timeout automatisch op 0 ingesteld.

Appendix F geeft meer informatie over communicatiefaciliteiten.

Voorbeeld 1:
```
10 OPEN "COM1:" AS 1
```

Bestand 1 wordt geopend voor communicatie, zonder waarden voor argumenten op te geven, die daarom op de standaardwaarden worden ingesteld: baud rate 300, even pariteit, 7 gegevens bits en 1 stop-bit.

Voorbeeld 2:
```
20 OPEN "COM1:2400" AS #2
```

Bestand 2 wordt geopend voor communicatie, met een baud rate van 2400 en de overige parameters op de standaardwaarde.

Voorbeeld 3:

```
30 OPEN "COM1:1200,N,8" AS #1
```

Bestand 1 wordt geopend voor asynchrone communicatie met 1200 bits per seconde en er wordt geen enkele vorm van pariteit gehanteerd of gecontroleerd.

OPTION BASE opdracht

Functie:

Instellen van de minimale waarde voor elke dimensie van een matrix.

Syntaxis:

OPTION BASE *n*

Opmerkingen:

Argument *n* kan alleen de waarde 0 en 1 hebben. De standaardwaarde is 0.

Als via OPTION BASE de waarde 1 wordt ingesteld, kan elke matrix-dimensie als laagste waarde 1 hebben.

Een negatieve waarde voor een matrix-dimensie is niet toegestaan.

OPTION BASE geeft alleen een foutmelding als u de waarde verandert. Zo is het mogelijk OPTION BASE-opdrachten op te nemen in aan elkaar gekoppelde programma's, zolang de begininstelling ongewijzigd blijft.

Attentie:

Als u probeert de waarde van OPTION BASE te wijzigen, nadat in het programma al matrices zijn aangemaakt, treedt een fout op.

OUT opdracht

Functie:

Een byte versturen naar de uitvoer-poort.

Syntaxis:

OUT *h,j*

Opmerkingen:

Zowel *h* als *j* zijn gehele getallen. Argument *h* verwijst naar het poortnummer, *j* naar de te verzenden gegevens. Voor *h* zijn getallen uit het bereik 0 t/m 65535 toegestaan, voor *j* getallen uit het bereik 0 t/m 255.

OUT voert de bewerking uit die het omgekeerde is van INP.

Voorbeeld:

```
100 OUT 12345,255
```

De decimale waarde 255 wordt verstuurd naar poortnummer 12345. Dezelfde instructie zou in assembleertaal als volgt luiden:

```
MOV DX,12345
MOV AL,255
OUT DX,AL
```

PAINT opdracht

Functie:

Een oppervlak inkleuren met een opgegeven patroon of kleur.

Syntaxis:

PAINT (*xstart*,*ystart*)[,*invul-kenmerk*[*kader*][,*achtergrond*]]

Opmerkingen:

De opdracht PAINT vult het vlak in een opgegeven kader in met het opgegeven *invul-kenmerk*. Als dat laatste kenmerk niet wordt opgegeven, wordt daarvoor het standaard voorgrondkenmerk (3 of 1) gehanteerd. De standaardwaarde voor *kader* is gelijk aan de waarde voor het *invul-kenmerk*. Zie de beschrijving van de opdrachten COLOR en PALETTE voor meer bijzonderheden.

PAINT moet beginnen vanaf een punt dat geen deel uitmaakt van het kader zelf, anders heeft PAINT geen effect.

Met PAINT kunt u elke afbeelding invullen, maar het maken van randen met uitsteeksels of zeer complexe tekeningen kan leiden tot een tekort aan geheugenruimte. Met behulp van een CLEAR-commando kan de ruimte op de stack worden vergroot.

Punten die buiten het gedefinieerde beeldscherm liggen, worden niet afgebeeld en er verschijnt in dat geval geen foutmelding. Raadpleeg de beschrijving van SCREEN voor de verschillende scherm-modi.

Paint tiling:

Het gebruik van inkleuren bij tegels ("tiles") lijkt op het kiezen van een stijl in de LINE-opdracht. Steeds wanneer een punt op het scherm wordt weergegeven, raadpleegt PAINT een tegelpatroon.

Als voor het *invul-kenmerk* geen waarde wordt opgegeven, gebruikt het systeem het standaardkenmerk voor de voorgrond.

Als het *invul-kenmerk* bestaat uit een numerieke expressie, moet de uitkomst een geldig kleurnummer zijn. Die waarde wordt dan gebruikt voor het inkleuren van het oppervlak.

Het tegelpatroon is altijd 8 bits breed en kan tussen 1 en 64 bytes lang zijn. Elke byte in de tile-reeks maskeert tijdens het plotten acht bits langs de x-as. Elk byte in de tile-reeks wordt geroteerd voor de aanpassing aan de y-as, waarbij geldt:

tile-byte-masker = y MOD tile-lengte

waarbij *y* de positie van de grafische cursor op de *y*-as aangeeft.

De term *tile-lengte* verwijst naar de lengte (in bytes) van de tile-reeks, zoals die is gedefinieerd door de gebruiker (tussen 1 en 64 bytes).

Deze bewerking wordt zodanig uitgevoerd dat het tegelpatroon over het gehele scherm gelijkmatig wordt weergegeven (zoals ook zou worden bereikt door PAINT(0,0)).

```
x neemt toe --> bit van tile-byte
x,y     8 7 6 5 4 3 2 1

0,0    |x|x|x|x|x|x|x|x|   tile byte 1
0,1    |x|x|x|x|x|x|x|x|   tile byte 2
0,2    |x|x|x|x|x|x|x|x|   tile byte 3
.
.
0,63   |x|x|x|x|x|x|x|x|   tile byte 64
                           (max. waarde)
```

Tijdens scherm-modi met hoge resolutie (SCREEN 2) kunt u via de volgende opdracht het gehele scherm laten opvullen met de letter X:

```
PAINT(320,100),CHR$(&H81)+CHR$(&H42)+CHR$(&H24)
+CHR$(&H18)+CHR$(&H18)+CHR$(&H24)+CHR$(&H42)+CHR$
(&H81)
```

Op het scherm verschijnt dit als volgt:

```
x neemt toe -->

0,0 |x| | | | | | |x|     CHR$(&H81)  tile-byte 1
0,1 | |x| | | | |x| |     CHR$(&H42)  tile-byte 2
0,2 | | |x| | |x| | |     CHR$(&H24)  tile-byte 3
0,3 | | | |x|x| | | |     CHR$(&H18)  tile-byte 4
0,4 | | | |x|x| | | |     CHR$(&H18)  tile-byte 5
0,5 | | |x| | |x| | |     CHR$(&H24)  tile-byte 6
0,6 | |x| | | | |x| |     CHR$(&H42)  tile-byte 7
0,7 |x| | | | | | |x|     CHR$(&H81)  tile-byte 8
```

Bij medium resolutie (SCREEN 1) worden voor elk beeldpunt twee bits gebruikt en daarom beschrijft elke byte van het tegelpatroon slechts vier pixels (beeldpunten). In dat geval beschrijven elke twee bits van de tile byte een van de vier mogelijke kleuren voor elk van de vier beeldpunten die moeten worden weergegeven.

Het argument *achtergrond* bepaalt de achtergrond van het tegelpatroon of de kleuren-byte die moet worden overgeslagen tijdens de controle of wanneer het einde van het kader is bereikt. Het argument *achtergrond* moet bestaan uit een tekenreeks-formule die leidt tot één teken. Als u zelf niets opgeeft, geldt de standaard tekenreeks CHR$(0).

Stel dat u een tegelpatroon over een reeds ingekleurd oppervlak heen wilt leggen, dat dezelfde kleur heeft als twee opeenvolgende regels in het tegelpatroon. De PAINT-bewerking wordt afgebroken als twee opeenvolgende regels dezelfde kleur hebben als het weer te geven beeldpunt. Zonder het argument *achtergrond* is het onmogelijk afwisselend rode en blauwe lijnen te tekenen op een rode achtergrond. PAINT onderbreekt de bewerking zodra de eerste rode pixel wordt getekend. Door voor de achtergrond rood (CHR$(&HAA)) op te geven, tekent de opdracht een rode lijn over de rode achtergrond.

U kunt in de tile-reeks maximaal twee opeenvolgende bytes opgeven die identiek zijn aan het argument *achtergrond*, bij overschrijding daarvan volgt een foutmelding.

Voorbeeld:

```
10 CLS
20 SCREEN 1
30 LINE (0,0)-(100,150),2,B
40 PAINT (50,50),1,2
50 LOCATE 20,1
```

De opdracht PAINT op regel 40 kleurt de op regel 30 getekende rechthoek in met kleur 1.

PALETTE, PALETTE USING opdrachten

Functie:

Een of meer kleuren van het palet wijzigen.

Syntaxis:

PALETTE [*kenmerk*,*kleur*]
PALETTE USING *matrix-naam* (*matrix-dimensie*)

Opmerkingen:

De opdracht PALETTE functioneert *alleen* bij systemen die zijn uitgerust met een Enhanced Graphics Adapter (EGA), of een met EGA compatibele video-adapter. Een GW-BASIC kleurenpalet bevat een set kleuren, waarbij elke kleur wordt gespecificeerd door een *kenmerk*. Elk *kenmerk* is gekoppeld aan een werkelijke *kleur* op het beeldscherm, die afhankelijk is van de ingestelde scherm-modus en de eigenschappen van de monitor.

PALETTE zonder toegevoegde argumenten stelt het kleurenpalet in op een bekende begininstelling, die overeenkomt met de instelling wanneer de kleuren de eerste keer worden geïnitialiseerd.

Bij toevoeging van argumenten verschijnt *kleur* zodra in een opdracht voor het bepalen van kleur het *kenmerk-argument* wordt gedefinieerd. Wijzigingen in de kleurspecificaties worden onmiddellijk op het scherm zichtbaar. Wanneer kleur-argumenten in grafische opdrachten worden gebruikt, refereren deze aan kenmerken, niet aan werkelijke kleuren. PALETTE koppelt kenmerken aan werkelijke kleuren.

Stel dat het huidige palet bestaat uit de *kleuren* 0, 1, 2 en 3.
De DRAW-opdracht

```
DRAW "C3L100"
```

kiest kenmerk 3 en tekent een lijn van 100 pixels in de kleur die bij kenmerk 3 hoort, in dit geval kleur 3. Bij uitvoering van de opdracht

```
PALETTE 3,2
```

wordt de kleur die bij kenmerk 3 hoort, veranderd in kleur 2. Alle informatie die tot dan toe via kenmerk 3 op het scherm werd weergegeven, verandert direct naar kleur 2. Alle tekst en grafische patronen die vanaf dat moment met behulp van kenmerk 3 worden weergegeven, zullen in kleur 2 verschijnen. Het aangepaste palet bevat de *kleuren* 0, 1, 2 en 2.

Met de optionele toevoeging USING kan de gehele inhoud van het palet met één PALETTE-opdracht worden gewijzigd. Het argument *matrix-naam* verwijst naar een matrix. *Matrix-dimensie* duidt op het nummer van het eerste matrix-element dat wordt gebruikt voor het instellen van het palet. Aan elk *kenmerk* in het palet wordt een daarmee corresponderende *kleur* uit de matrix toegekend.

De afmeting van de matrix moet groot genoeg zijn om alle achter *matrix-dimensie* in het palet ingevoerde gegevens te kunnen onderbrengen. Stel dat u aan alle 16 kenmerken kleuren toewijst en dat u in de PALETTE USING-opdracht bij de matrix-dimensie het nummer 5 gebruikt als laagste waarde. Dan moet de matrix groot genoeg zijn voor de opslag van minimaal 20 elementen (want het bereik 5 t/m 20 omvat 16 elementen).

```
DIM PAL%(20)
.
.
.
PALETTE USING PAL%(5)
```

Als de waarde voor het argument *kleur* in een matrix gelijk is aan -1, dan wordt het bijbehorende *kenmerk* niet gewijzigd. Voor het argument *kleur* zijn geen andere negatieve waarden toegestaan.

Het is mogelijk het argument *kleur* te gebruiken om in een COLOR-opdracht de standaardkleur voor tekstweergave in te stellen. (Vergeet niet dat de kleurargumenten in andere BASIC-instructies overeenkomen met wat hier "*kenmerk*" wordt genoemd.) Dit kleur-argument bepaalt hoe tekst op het scherm wordt weergegeven. Bij een normale basisinstelling van het palet, resulteert *kenmerk* 0 in zwart. Met PALETTE kunt u, bijvoorbeeld, bereiken dat *kenmerk* 0 wordt gekoppeld aan wit in plaats van zwart.

We vermelden nogmaals dat een PALETTE-opdracht zonder toegevoegde argumenten alle *kenmerken* instelt op de standaard *kleuren*.

Onderstaande tabel geeft voor verschillende scherm-modi en monitor-types een overzicht van de mogelijke waarden voor de argumenten *kenmerk* en *kleur*.

Scherm-modus	Monitor-type	Adapter	Waarden voor kenmerk	Waarden voor kleur
0	monochroom	MDPA	n.v.t.	n.v.t.
	monochroom	EGA	0-15	0-2
	kleur	CGA	n.v.t.	0-31[a]
	kleur/Enh.[d]	EGA	0-31[a]	0-15
1	kleur	CGA	n.v.t.	0-3
	kleur/Enh.[d]	EGA	0-3	0-15
2	kleur	CGA	n.v.t.	0-1
	kleur/Enh.[d]	EGA	0-1	0-15
7	kleur/Enh.[d]	EGA	0-15	0-15
8	kleur/Enh.[d]	EGA	0-15	0-15
9	Enhanced [d]	EGA[b]	0-3	0-15
	Enhanced [d]	EGA[c]	0-15	0-63
10	monochroom	EGA	0-3	0-8

a = kenmerken 16-31 voor knipperen van kleuren 0-15
b = 64K EGA-geheugen
c = meer dan 64K EGA-geheugen
d = Enhanced Color Display
Enh. = Enhanced
CGA = Color Graphics Adapter en CGA-compatibele adapters
EGA = Enhanced Graphics Adapter en EGA-compatibele adapters
MDPA = IBM Monochrome Display and Printer Adapter

Raadpleeg bij de SCREEN-opdracht de lijst met kleuren die beschikbaar zijn voor de verschillende combinaties van scherm-modi, monitor en video-adapter.

Voorbeelden:

```
PALETTE 0,2              'Wijzigt de kleur van
                         'kenmerk 0 naar kleur 2

PALETTE 0,-1             'Palet blijft ongewijzigd

PALETTE USING A%(0)      'Wijzigt elk gegeven in het
                         'palet. Omdat in alle cellen
                         'van een matrix bij het ini-
                         'tialiseren de waarde 0
                         'wordt gezet, worden alle
                         'kenmerken nu ingesteld op
                         'het weergeven van kleur 0.
                         'Op het scherm verschijnt
                         'één kleur. Het blijft
                         'mogelijk BASIC instructies
                         'uit te voeren.

PALETTE                  'Elk gegeven in het palet
                         'wordt ingesteld op de kleur
                         'die bij het initialiseren
                         'wordt gekozen. Hoe die
                         'kleuren werkelijk op het
                         'scherm verschijnen, is
                         'afhankelijk van het type
                         'monitor.
```

PCOPY commando

Functie:
Een schermpagina kopiëren naar een andere, tijdens alle scherm-modi.

Syntaxis:
PCOPY *bronpagina, bestemmingspagina*

Opmerkingen:
Het argument *bronpagina* moet bestaan uit een geheel getal uit het bereik 0 tot *n*, waarbij *n* afhankelijk is van de huidige afmeting van het video-geheugen en de hoeveelheid geheugen die door elke afzonderlijke pagina in beslag wordt genomen.

Voor de *bestemmingspagina*, de pagina waarnaar de *bronpagina* moet worden gekopieerd, gelden dezelfde vereisten.

Raadpleeg de beschrijving van CLEAR en SCREEN voor meer bijzonderheden.

Voorbeeld:

```
PCOPY 1,2
```

Deze opdracht kopieert de inhoud van schermpagina 1 naar schermpagina 2.

PEEK functie

Functie:

De inhoud van een opgegeven geheugenadres lezen.

Syntaxis:

PEEK(*a*)

Opmerkingen:

Geeft de decimale gehele waarde (0-255) van de byte die is opgeslagen in geheugenadres *a*. Voor *a* moet een waarde worden opgegeven uit het bereik 0 t/m 65535.

De laatst uitgevoerde opdracht DEF SEG bepaalt het absolute adres waarvan de inhoud wordt gelezen.

PEEK voert de omgekeerde bewerking uit van POKE.

Voorbeeld:

```
10 A=PEEK(&H5A00)
```

In variabele A wordt de waarde opgeslagen die overeenkomt met de inhoud van het geheugenadres hex 5A00 (gelijk aan 23040 decimaal). Vergelijk dit met het voorbeeld bij POKE.

PEN opdracht en functie

Functie:

Gebruik van de lichtpen detecteren en positie ervan lezen.

Syntaxis:

Als opdracht:

PEN ON
PEN OFF
PEN STOP

Als functie:

x = P(*n*)

Opmerkingen:

De variabele *x* dient voor de opslag van de PEN-positie.

Voor *n* moet een geheel getal uit het bereik 0-9 worden gebruikt.

PEN ON activeert het inlezen van de PEN-positie.

PEN OFF schakelt het inlezen uit.

PEN STOP detecteert wel het gebruik van de pen, maar reageert daar pas op na de eerstvolgende PEN ON-opdracht.

x=PEN(n) leest de coördinaten van de penpositie.

De PEN-functie wordt niet automatisch geactiveerd. Daarom moet voor het detecteren van de penpositie een PEN ON-opdracht worden gebruikt voordat de PEN-functie de penpositie kan lezen. Zonder voorafgaande PEN ON-opdracht leidt de functie x=PEN(n) tot de foutmelding "Onjuiste functieaanroep".

Coördinaten van lichtpen:

n=0 Geeft -1 als de penschakelaar werd ingedrukt nadat de laatste keer de pencoördinaten werden opgevraagd. In andere gevallen geeft n=0 de waarde 0.

n=1 Geeft de x-coördinaat van de pixel waar de pen voor het laatst werd geactiveerd. Deze waarde voor de horizontale coördinaat is een waarde tussen 0 en 319 bij medium resolutie en tussen 0 en 639 bij hoge resolutie.

n=2 Geeft de y-coördinaat van de pixel waar de pen voor het laatst werd geactiveerd; een waarde uit het bereik 0-199.

n=3 Geeft de huidige status van de penschakelaar. De waarde -1 duidt op ingedrukt en 0 duidt op niet ingedrukt.

n=4 Geeft de laatst gedetecteerde geldige x-coördinaat. Bij medium resolutie is dat een waarde tussen 0 en 319, bij hoge resolutie een waarde tussen 0 en 639.

n=5 Geeft de laatst gedetecteerde geldige y-coördinaat; een waarde tussen 0 en 199.

n=6 Geeft de rij-coördinaat van het teken waarbij de pen voor het laatst werd geactiveerd.

n=7 Geeft de kolom-coördinaat van het teken waarbij de pen voor het laatst werd geactiveerd. De waarde van die coördinaat ligt in het bereik 1-40 of 1-80, afhankelijk van de ingestelde schermbreedte.

n=8 Geeft de laatst gedetecteerde geldige schermregel; een waarde uit het bereik 1-24.

n=9 Geeft de laatst gedetecteerde geldige tekenkolom (=positie op de schermregel); een waarde uit het bereik 1-40 of 1-80, afhankelijk van de ingestelde schermbreedte.

Ter verbetering van de verwerkingssnelheid is het raadzaam de detectie van de lichtpen uit te schakelen (met PEN OFF) als in een programma de lichtpen niet wordt gebruikt.

Als de pen zich dicht tegen de rand van het scherm bevindt, zijn de aangegeven coördinaten van de penpositie soms onnauwkeurig.

Voorbeeld:

```
50 PEN ON
60 FOR I=1 TO 500
70 X=PEN(0): X1=PEN(3)
80 PRINT X,X1
90 NEXT
100 PEN OFF
```

Dit voorbeeldprogramma geeft de vorige status en de huidige status na verplaatsing van de lichtpen.

PLAY opdracht

Functie:

Muziek weergeven door muziek-macro's onder te brengen in tekenreeksen.

Syntaxis:

PLAY *tekenreeks-expressie*

Opmerkingen:

De argumenten voor PLAY bestaan uit één teken en luiden als volgt:

A-G[#,+,-]	A-G zijn noten. Een hekje (#) of een plusteken (+) na de noot resulteren in een verhoogde noot, een minteken (-) leidt tot een verlaagde noot. Elke noot gevolgd door #, + of - moet verwijzen naar een toets die op een piano zwart is.
L(*n*)	Bepaalt de lengte van elke toon. L4 is een kwartnoot, L1 een hele noot, enz. De waarde van *n* mag variëren tussen 1 en 64. De lengte mag worden opgegeven achter een noot, om alleen de lengte van die ene toon te wijzigen. A16 is gelijk aan L16A.
MF	Music Foreground, ofwel voorgrondmuziek. PLAY- en SOUND-opdrachten worden als voorgrond-instructies beschouwd. Dit houdt in dat elke toon volledig uitsterft alvorens de volgende begint. Dit is na het opstarten de standaardinstelling.
MB	Music Background, ofwel achtergrondmuziek. PLAY en SOUND worden als achtergrond-instructies beschouwd. Dit betekent dat elke noot of elk geluid wordt opgeslagen in een buffer, zodat de uitvoering van het BASIC programma doorgaat terwijl "op de achtergrond" muziek weerklinkt. Via MB kunnen 32 noten tegelijkertijd op de achtergrond worden afgespeeld.
MN	Music Normal, ofwel normale muziekweergave. Elke noot houdt gedurende 7/8 deel van de via L opgegeven tijdsduur aan.
ML	Music Legato (gebonden). Elke noot houdt de gehele door L bepaalde tijdsduur aan.

MS	Music Staccato. Elke noot klinkt afgebroken, duurt slechts 3/4 van de door L bepaalde tijd.
N(*n*)	Geeft noot *n* weer, waarbij *n* een waarde uit het bereik 0 tot 84 kan hebben. In de beschikbare octaven bevinden zich 84 noten. Als *n* gelijk is aan 0, resulteert dat in een pauze.
O(*n*)	O staat voor "octaaf". Octaaf 0 is het huidige octaaf. Er zijn 7 octaven, namelijk 0 t/m 6. De standaardwaarde is 4. De midden-C ligt aan het begin van octaaf 3.
P(*n*)	Pauze, waarbij *n* kan variëren van 1 t/m 64.
T(*n*)	Tempo. T bepaalt het aantal L4 (kwartnoten) per minuut. De waarde van *n* mag variëren van 32 t/m 255. De standaardwaarde is 120.
. (punt)	Een punt na een noot verlengt de tijdsduur (L) van een noot met 0,5 maal de waarde van T (Tempo). Achter een noot kunnen meerdere punten staan, waaraan de lengte van de noten zich aanpast. Zo zal A. de noot A 1,5 maal de lengte van L (Lengte) maal T (Tempo) laten duren. A.. geeft een noot A met een tijdsduur van 9/4 maal L, enz. Ook achter de P (Pauze) mag een punt staan, die de tijdsduur van de pauze overeenkomstig de boven omschreven regels zal doen toenemen.
Xreeks;	Voert een subreeks uit, waarbij de reeks een variabele is die in een aantal PLAY-commando's voorkomt. Sommige noten werken niet bij een hoog tempo, wat te wijten is aan de lage interrupt-snelheid van de interne klok. Probeer maar eens 1.64 bij T255. Deze combinaties moet u door uitproberen onderzoeken.
>*n*	Het symbool "groter dan" voorafgaand aan noot *n* zorgt dat de noot een octaaf hoger wordt afgespeeld.
<*n*	Het symbool "kleiner dan" voorafgaand aan noot *n* zorgt dat de noot een octaaf lager wordt afgespeeld.

N.B.:

Numerieke argumenten voldoen aan de syntaxis-eisen die zijn omschreven bij de opdracht DRAW.

Het argument *n* mag een variabele zijn, voorafgegaan door het is-gelijk teken (=), maar ook een constante. De variabele en ook de variabele in Xreeks moeten worden gevolgd door een puntkomma.

PLAY(n) functie

Functie:

Het aantal noten aangeven dat in de wachtrij voor achtergrondmuziek staat.

Syntaxis:
PLAY(*n*)

Opmerkingen:
Argument *n* is een zogenaamd "dummy-argument", en mag dus een willekeurige waarde hebben.

PLAY(*n*) geeft de waarde 0 bij voorgrondmuziek.

PLAY(*n*) leidt nooit tot een uitkomst hoger dan 32.

Voorbeeld:
```
10 REM Als de wachtrij nog vier noten bevat,
20 'speel dan een andere melodie.
30 PLAY "MBABCDABCDABCD"
40 IF PLAY(0)=4 THEN 200
.
.
.
200 PLAY "MBCDEFCDEF"
```

PMAP functie (Grafisch)

Functie:
Expressies vertalen naar logische of fysieke coördinaten.

Syntaxis:
x = PMAP (*exp,functie*)

Opmerkingen:
Deze functie werkt alleen tijdens de grafische modi.

De waarde van *x* is de fysieke coördinaat van het punt dat moet worden vertaald.

Argument *exp* bestaat uit een numerieke variabele of expressie.

Functie	Omzetting
0	logische expressie naar fysieke x
1	logische expressie naar fysieke y
2	fysieke expressie naar logische x
3	fysieke expressie naar logische y

PMAP wordt bij VIEW en WINDOW gebruikt om coördinaten te vertalen.

POINT functie

Functie:
Aflezen van de kleur- of kenmerk-waarde van een beeldpunt op het scherm.

Syntaxis:
POINT (*x,y*)
POINT (*functie*)

Opmerkingen:

In de eerste vorm van de syntaxis zijn *x* en *y* de coördinaten van het onderzochte beeldpunt. Als daarvoor een waarde wordt opgegeven die niet binnen het geldige bereik ligt, leidt POINT tot de uitkomst -1.

Raadpleeg de beschrijving van COLOR en PALETTE voor geldige waarden voor kenmerken en kleuren.

Als u slechts één argument gebruikt, kunt u de huidige grafische coördinaten opvragen.

De tweede syntaxis, POINT (*functie*), geeft de huidige grafische x- en y-coördinaten, waarbij geldt:

Functie	Resultaat
0	huidige fysieke x-coördinaat
1	huidige fysieke y-coördinaat
2	huidige logische x-coördinaat als een WINDOW-opdracht geactiveerd is; anders de huidige fysieke x-coördinaat zoals bij functie = 0
3	huidige logische y-coördinaat als een WINDOW-opdracht geactiveerd is; anders de huidige fysieke y-coördinaat zoals bij functie = 1

Voorbeeld 1:

```
10 SCREEN 1
20 FOR C = 0 TO 3
30 PSET (10,10),C
40 IF POINT(10,10)<>C THEN PRINT "Defecte BASIC="
50 NEXT C
RUN
```

Voorbeeld 2:

```
10 SCREEN 2
20 IF POINT(I,I)<>0 THEN PRESET (I,I) ELSE PSET (I,I)
RUN
```

Dit voorbeeld keert de huidige status van een beeldpunt om.

Voorbeeld 3:

```
20 PSET(I,I),1-POINT(I,I)
RUN
```

Dit is een ander methode voor het omkeren van de status.

POKE opdracht

Functie:

Een waarde (één byte) opslaan in een opgegeven geheugenadres.

Syntaxis:

POKE *a,b*

Opmerkingen:

Zowel *a* als *b* moeten gehele getallen zijn.

De waarde voor *a* bepaalt het geheugenadres waarin de gegevens moeten worden weggeschreven ("ge-poked"). De laatst uitgevoerde DEF SEG-opdracht bepaalt het geheugenadres. Opgegeven geheugenadressen worden door GW-BASIC niet gecontroleerd. Voor *a* moet een geheel getal uit het bereik 0 t/m 65535 worden opgegeven.

Argument *b* is het gegeven dat in geheugenadres *a* moet worden opgeslagen. Voor *b* moet een geheel getal uit het bereik 0 t/m 255 worden opgegeven.

PEEK is de omgekeerde bewerking van POKE. Met PEEK kunt u de inhoud van een geheugenadres lezen.

POKE en PEEK zijn nuttig voor het efficiënt opslaan van gegevens, voor het laden van assembleerroutines en voor het uitwisselen van gegevens tussen dergelijke routines en BASIC programma's.

Voorbeeld:

```
10 POKE &HA500,&HFF
```

Deze instructie slaat de decimale waarde 255 (hexadecimaal FF) op in geheugenadres 23040 (hexadecimaal 5A00). Vergelijk dit met het voorbeeld bij PEEK.

POS functie

Functie:
Aangeven van de huidige cursor-positie.

Syntaxis:
POS(*c*)

Opmerkingen:
De meest linkse positie is 1.

Argument *c* is een zogenaamd "dummy-argument", waarvoor u een willekeurige waarde mag opgeven.

Voorbeeld:

```
10 CLS
20 WIDTH 80
30 A$=INKEY$: IF A$="" THEN GOTO 30 ELSE PRINT A$;
40 IF POS(X)>10 THEN PRINT CHR$(13);
50 GOTO 30
```

Dit programma zorgt dat na weergave van het tiende teken op een schermregel de cursor naar een nieuwe regel gaat (want CHR$(13) betekent een carriage return).

PRESET en PSET opdrachten

Functie:
In grafische modus een punt afbeelden op een bepaalde schermpositie.

Syntaxis:
PRESET(*x*,*y*)[,*kleur*]
PSET(*x*,*y*)[,*kleur*]

Opmerkingen:

Met (*x*,*y*) bepaalt u de coördinaten van de schermpositie.

Het argument kleur bepaalt de weergavekleur.

De coördinaten kunnen als absolute of als relatieve waarden worden opgegeven.

Absolute coördinaten:

De vorm (*absoluut x*,*absoluut y*) wordt veel gebruikt. De coördinaten verwijzen dan direct naar een beeldpunt zonder rekening te houden met de laatst gebruikte beeldpositie. Bijvoorbeeld:

```
(10,10)
```

Relatieve coördinaten:

STEP (*x beginpositie*,*y beginpositie*) verwijst naar een punt op een bepaalde horizontale en verticale afstand van de laatste grafische schermpositie. Bijvoorbeeld:

```
STEP(10,10)
```

De waarden voor coördinaten kunnen verwijzen naar een positie die momenteel buiten het beeldscherm ligt. De waarden voor de coördinaten moeten echter steeds tussen de gehele getallen -32768 en +32767 liggen, anders verschijnt de foutmelding "Te groot getal".

De waarde (0,0) verwijst altijd naar de linkerbovenhoek van het scherm en (0,199) naar de hoek linksonder. Dit geldt zowel voor medium als hoog oplossend vermogen.

Raadpleeg de opdrachten COLOR en PALETTE voor meer bijzonderheden.

Als u voor kleur een waarde van 4 of groter opgeeft, treedt de fout "Onjuiste functieaanroep" op.

Voorbeeld 1:

```
10 CLS
20 SCREEN 1
30 FOR P=0 TO 100
40 PSET (P,P)
50 NEXT
60 LOCATE 14,1
```

Dit programma tekent een diagonale lijn tussen de coördinaten (0,0) en (100,100).

Voorbeeld 2:

```
40 FOR P=100 TO 0 STEP -1
50 PSET(P,P),0
60 NEXT P
```

Wist de lijn door elk beeldpunt op nul te zetten.

PRINT opdracht

Functie:
Gegevens op het scherm afbeelden.

Syntaxis:
PRINT [*expressielijst*][;]
? [*expressielijst*][;]

Opmerkingen:
Als u achter PRINT geen *expressies* aangeeft, verschijnt een lege regel. Als u een of meer expressies opgeeft, worden de waarden daarvan op het scherm afgebeeld. De expressies mogen zowel numerieke als tekenreeks-expressies zijn. De expressies of variabelenamen moeten onderling worden gescheiden door komma's, spaties of puntkomma's. Letterlijk af te beelden tekst moet tussen aanhalingstekens (") staan.

Raadpleeg de beschrijving van STRING$ voor meer bijzonderheden over tekenreeksen.

Binnen GW-BASIC kunt u het woord "PRINT" vervangen door een vraagteken (?).

Indeling van een schermregel:

GW-BASIC deelt een schermregel op in afzonderlijke zones die elk 14 tekenposities lang zijn. De scheidingstekens in de PRINT-opdracht bepalen op welke regelpositie een gegeven wordt afgebeeld:

Scheidingsteken	Regelpositie
,	Begin van volgende zone
;	Onmiddellijk achter laatst afgebeelde gegeven
Spatie(s)	Onmiddellijk achter laatst afgebeelde gegeven

Als de expressielijst wordt afgesloten door een komma, puntkomma of de functies SPC of TAB, dan wordt de expressie uit de eerstvolgende PRINT-opdracht op dezelfde regel afgebeeld, van het voorgaande gegeven gescheiden door spaties. Als na de expressielijst geen SPC- of TAB-functie, komma of puntkomma staat, voegt het systeem automatisch een carriage return toe na het laatst afgebeelde gegeven. (De cursor gaat dan automatisch naar de eerstvolgende schermregel.)

Na het weergeven van dubbelbrede tekens voegt GW-BASIC automatisch een carriage return/line feed toe zodra 40 of 80 tekens zijn afgebeeld, afhankelijk van de waarde die in een WIDTH-opdracht is opgegeven. Als u precies 40, respectievelijk 80 tekens laat afbeelden, slaat de cursor na het afbeelden van de expressie(s) twee regels over, behalve wanneer u de expressielijst hebt afgesloten met een puntkomma.

Als u getallen op het scherm laat afbeelden, wordt na elk getal altijd een spatie afgebeeld. Positieve getallen worden voorafgegaan door een spatie en negatieve getallen door een minteken (-). Getallen met enkele precisie verschijnen met 7 of minder cijfers in de formats integer (gehele getallen) of fixed-point (getallen met een decimale punt).

Lees ook de beschrijving van LPRINT en LPRINT USING voor het afdrukken van gegevens via de printer.

Voorbeeld:

```
10 X$=STRING$(10,45)
20 PRINT X$"Maandoverzicht" X$
RUN
----------Maandoverzicht----------
Ok
```

De waarde 10 bepaalt hoe vaak het teken met ASCII-code 45 (het minteken) wordt afgedrukt.

PRINT USING opdracht

Functie:

Gegevens afbeelden volgens een opgegeven layout.

Syntaxis:

PRINT USING *tekenreeks-expressie;expressielijst*

Opmerkingen:

Het argument *tekenreeks-expressie* (ook wel "format-string" genoemd) bestaat uit letterlijke tekenreeks tussen aanhalingstekens of uit een variabele waarin speciale tekens voor de layout zijn opgeslagen. Deze tekens bepalen in welke vorm en op welke positie de tekenreeksen of getallen worden afgebeeld.

In het argument *expressielijst* geeft u de tekenreeksen of numerieke gegevens op die op het scherm moeten verschijnen. De gegevens in de lijst moeten onderling worden gescheiden door puntkomma's.

Tekenreeks-velden:

Onderstaande drie tekens dienen voor het bepalen van de vorm van een tekenreeks-veld:

!	Geeft aan dat van de tekenreeks alleen het eerste teken moet worden afgebeeld.
n spaties\	Geeft aan dat van de tekenreeks 2 + *n* tekens moeten worden afgebeeld. Als u beide backslashes (\) zonder tussenliggende spaties gebruikt, worden twee tekens van de tekenreeks afgebeeld, bij twee backslashes met één spatie verschijnen drie tekens, enz. Als de tekenreeks langer is dan het opgegeven aantal tekens, negeert GW-BASIC de resterende tekens. Als de tekenreeks korter is dan de opgegeven lengte, dan verschijnen de tekens links in het veld en worden de resterende veldposities opgevuld met spaties.

Een voorbeeld:

```
10 A$="Kijk": B$="uit"
30 PRINT USING "!!";A$;B$
40 PRINT USING "\  \";A$;B$
30 PRINT USING "\   \";A$;B$;"!!"
RUN
Ku
Kijkuit
Kijk uit  !!
Ok
```

& Past de lengte van het tekenreeks-veld aan aan de lengte van de afgebeelde tekenreeks, die dus precies zo op het scherm verschijnt als hij wordt ingetikt. Bijvoorbeeld:

```
10 A$="Kijk":B$="uit"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
RUN
Kuit
Ok
```

Numerieke velden:

De volgende speciale tekens dienen voor het bepalen van de layout van numerieke velden:

Voor elke cijferpositie in een numeriek veld gebruikt u een hekje. De op die wijze gereserveerde posities worden altijd gevuld. Als het getal minder cijfers telt dan het aantal gereserveerde posities, dan verschijnen voor het afgebeelde getal een of meer spaties.

U kunt de decimale punt op een willekeurige positie in het veld invoegen. Elk cijfer vóór de positie van de decimale punt wordt afgebeeld, ook een 0. Indien noodzakelijk rondt GW-BASIC getallen eerst af. Enkele voorbeelden:

```
PRINT USING "##.##";.78
0.78

PRINT USING "###.##";987.654
987.65

PRINT USING "##.##  "; 10.2,5.3,
66.789,.234
10.20    5.30    66.79    0.23
```

In het laatste voorbeeld eindigt de layout-tekenreeks op drie spaties, zodat de getallen keurig van elkaar gescheiden op het scherm verschijnen.

+ Een plusteken aan het begin of einde van een layout-tekenreeks zorgt dat vóór, respectievelijk achter het getal een plus- of minteken verschijnt. Bijvoorbeeld:

```
PRINT USING"+##.## ";-68.95,2.4,55.6,
-.9,-68,95
+2.40 +55.60  -0.90
ok
```

`-` Als u de layout-tekenreeks afsluit met een minteken, dan verschijnt achter negatieve getallen een minteken. Bijvoorbeeld:

```
PRINT USING"##.##- ";-68.95,22.449,
-7.01
68.95- 22.45    7.01-
ok
```

`**` Twee sterretjes aan het begin van de tekenreeks zorgen dat aan getallen voorafgaande spaties worden opgevuld met een sterretje. Bovendien zorgen zij voor twee extra cijferposities. Bijvoorbeeld:

```
PRINT USING "**#.#";12.39,-0.9,765.1
*12.4*-0.9765.1
```

`$$` Twee dollartekens aan het begin van de tekenreeks laten links van het getal een $ verschijnen en zorgen voor twee extra posities, waarvan 1 positie voor het dollarteken. Bij $$ kan geen exponentiële notatie worden gebruikt. Negatieve getallen zijn alleen mogelijk als het minteken achter het getal wordt gezet. Een voorbeeld:

```
PRINT USING "$$###.##";456.78
$456.78
```

`**$` Deze tekenreeks combineert de laatstgenoemde twee effecten. Spaties links van het getal worden opgevuld met sterretjes en vóór het getal verschijnt een $. Tevens zorgt deze tekenreeks voor drie extra cijferposities, waarvan 1 voor de $. Bijvoorbeeld:

```
PRINT USING "**$##.##";2.34
***$2.34
```

`,` Een komma direct voorafgaand aan de decimale punt in de tekenreeks, zorgt dat in grote getallen de duizendtal-komma's worden afgebeeld (het equivalent van de nederlandse duizendtal-punt). Als u de tekenreeks afsluit met een komma, verschijnt deze als deel van de tekenreeks. Een voorbeeld:

```
PRINT USING "####,.##";1234.5
1,234.50
```

`^^^^` Vier caret-tekens (^) achter de cijferposities in de layout-tekenreeks zorgen voor de exponentiële notatie. De vier carets bieden plaats aan de tekenreeks E+xx. U mag zelf bepalen waar u de decimale punt plaatst. De cijfers worden links aansluitend afgebeeld en direct daarachter de exponent. Als u de tekenreeks niet laat beginnen met een + of eindigen met + of -, dan wordt 1 positie links van de decimale punt gebruikt voor het afbeelden van een spatie of een minteken. Enkele voorbeelden:

```
PRINT USING "##.##^^^^";234.56
2.35E+02
```

```
PRINT USING ".####^^^^-";888888
.8889E+06

PRINT USING "+.##^^^^";123
+.12E+03
```

_ Een onderstreepteken (_) in de layout-tekenreeks zorgt dat het eerstvolgende teken precies wordt afgebeeld zoals het is ingetikt. Een voorbeeld:

```
PRINT USING "_!##.##_!";1234
!12.34!
```

Het letterlijk af te beelden teken mag zelf ook een onderstreepteken zijn.

% Als het af te beelden getal te lang is voor het aantal gereserveerde cijferposities, dan verschijnt links van het getal een percentteken. Dit gebeurt ook als een getal na afronding te lang is geworden. Twee voorbeelden:

```
PRINT USING "##.##";111.22
%111.22

PRINT USING ".##";.999
%1.00
```

Als voor het afbeelden van een getal meer dan 24 cijferposities noodzakelijk zijn, dan verschijnt de foutmelding "Onjuiste functieaanroep".

PRINT# en PRINT# USING opdrachten

Functie:
Gegevens opslaan in een sequentieel diskette-bestand

Syntaxis:
PRINT#*bestandsnummer*,[USING *tekenreeks-expressie*;]*expressielijst*

Opmerkingen:
Bij *bestandsnummer* geeft u het nummer op dat werd toegekend toen het bestand werd geopend.

In het argument *tekenreeks-expressie* kunt u layout-tekens opnemen, zoals eerder beschreven bij PRINT USING.

De *expressielijst* bestaat uit numerieke en/of tekenreeks-expressies die in het bestand moeten worden opgeslagen.

Aanhalingstekens (") fungeren als scheidingstekens tussen numerieke en/of tekenreeks-expressies. Het eerste aanhalingsteken markeert het begin van de gegevens en het tweede sluit de gegevens af.

Als een expressie precies moet worden afgebeeld zoals die is ingetikt, dan moet die expressie tussen aanhalingstekens staan. Worden deze achterwege gelaten, dan wordt de aan die numerieke of tekenreeks-expressie toegekende waarde afgebeeld. Als geen waarde is toegekend, wordt automatisch de waarde 0 gehanteerd. De aanhalingstekens verschijnen niet op het beeldscherm. Enkele voorbeelden:

```
10 PRINT#1,A
0

10 A=26
20 PRINT#1,A
RUN
26

10 A=26
20 PRINT#1,"A"
RUN
A
```

Als u binnen een tekenreeks zelf aanhalingstekens wilt gebruiken, vervangt u die door CHR$(34), de ASCII-code voor aanhalingstekens. Bijvoorbeeld:

```
PRINT#1,"Hij zei,"hallo", geloof ik."
Hij zei, 0, geloof ik.
```

Dit loopt dus mis omdat het systeem hallo als een variabelenaam leest, waarvoor vooraf geen waarde is opgegeven.

```
PRINT#1, "Hij zei "CHR$(34)"hallo"CHR$(34)",
geloof ik."
Hij zei "hallo", geloof ik.
```

Als in de tekenreeks komma's, puntkomma's of belangrijke spaties staan, omsluit deze dan door aanhalingstekens ("). In het volgende voorbeeld wordt in de variabele A$ de term "CAMERA, AUTOMATISCH" opgeslagen en in B$ de type-aanduiding "93604-1":

```
10 A$="CAMERA, AUTOMATISCH": B$="93604-1"
20 PRINT #1,A$;B$
30 INPUT#1,A$,B$
```

Om deze tekenreeksen ordelijk van elkaar te scheiden, gebruiken we weer CHR$(34) voor de aanhalingstekens ("):

```
40 PRINT#1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
RUN
"CAMERA, AUTOMATISCH""93604-1"
```

U kunt de opdracht PRINT# combineren met USING, om de gegevens volgens een bepaald format op te slaan. Een voorbeeld:

```
PRINT #1, USING"$$###.##.";J;K;L
```

PRINT# comprimeert de gegevens niet; het slaat de gegevens op zoals ze ook op het scherm zouden verschijnen via PRINT. Vergeet daarom niet de gegevens zorgvuldig van elkaar te scheiden, zodat ze later weer op de juiste wijze worden ingelezen. Gebruik daarvoor puntkomma's als scheidingsteken:

```
PRINT#1, A;B;C;D;X;Y;Z
```

Wanneer u komma's als scheidingstekens gebruikt, slaat het programma tussen de gegevens een aantal lege tekenposities op. Dit geldt echter niet voor het opslaan van gegevens in exponentiële notatie.

Voor het scheiden van tekenreeks-expressies gebruikt u puntkomma's. Om deze gegevens met het juiste format op te slaan, plaatst u in de expressielijst expliciete scheidingstekens, bijvoorbeeld:

```
10 A$="CAMERA":B$="93604-1"
20 PRINT#1,A$,B$
```

wordt als volgt opgeslagen:

```
CAMERA93604-1
```

Zou u dit later weer laden, dan wordt het als één gegeven beschouwd, omdat in de expressielijst geen scheidingstekens werden gebruikt. U kunt dit probleem voorkomen door expliciet scheidingstekens toe te passen:

```
30 PRINT#1,A$;",";B$
```

Dit wordt op diskette opgeslagen in een format dat later wel als twee afzonderlijke gegevens wordt beschouwd:

```
CAMERA,93604-1
```

PUT opdracht (Grafisch)

Functie:
Grafische afbeeldingen naar het beeldscherm sturen.

Syntaxis:
PUT(*x*,*y*)*matrix*,[,*instructie*]

Opmerkingen:
Voor *instructie* kunt u opgeven: PSET, PRESET, AND, OR en XOR.

Argument (*x*,*y*) bevat de coördinaten van de linkerbovenhoek van de afbeelding.

PUT en GET zorgen voor transport van grafische afbeeldingen van en naar het scherm. Deze opdrachten bieden binnen de grafische modi mogelijkheden tot animaties en het snel bewegen van objecten.

Met behulp van PUT stuurt u een in een matrix opgeslagen afbeelding naar het beeldscherm, waarbij de coördinaten van de linkerbovenhoek als referentiepunt dienen. Als de afbeelding te groot is om binnen de geactiveerde scherm-modus te worden afgebeeld, dan treedt de fout "Onjuiste functieaanroep" op.

Met het argument *instructie* kunt u de naar het scherm gestuurde afbeelding samenvoegen met de afbeelding die zich al op het scherm bevindt. PSET stuurt de gegevens ongewijzigd naar het scherm.

PRESET heeft dezelfde functie als PSET, maar zorgt voor een zwarte afbeelding op een witte achtergrond.

AND verstuurt de afbeelding alleen als er onder die afbeelding al een afbeelding bestaat.

OR projecteert de afbeelding op de bestaande afbeelding.

XOR wordt veel gebruikt voor animaties. Met behulp van XOR wordt elk beeldpunt geïnverteerd waarvoor ook in het matrix een beeldpunt is opgeslagen. Dit komt exact overeen met het gedrag dat de cursor op het scherm vertoont.

PUT opdracht (Bestanden)

Functie:

Een record vanuit een bestandsbuffer naar een onbepaald benaderbaar bestand schrijven.

Syntaxis:

PUT[#] *bestandsnummer*[,*record-nummer*]

Opmerkingen:

Bij *bestandsnummer* geeft u het nummer op dat werd toegekend toen het bestand werd geopend.

Het *record-nummer* is het nummer dat naar een bepaald record verwijst. Als u geen record-nummer opgeeft, wordt het record-nummer gekozen dat 1 hoger is dan het record-nummer dat in de laatst gebruikte PUT-opdracht werd gebruikt.

Met behulp van PRINT#, PRINT# USING, LSET, RSET of WRITE# kunt u tekens in de bestandsbuffer opslaan voordat u een PUT-opdracht gebruikt. Bij toepassing van WRITE# vult GW-BASIC de buffer met spaties tot de eerstvolgende return.

Elke poging voorbij het einde van de buffer te lezen of te schrijven, leidt tot de fout "Field-lengte te groot".

PUT is ook bruikbaar bij communicatie-bestanden. In dat geval duidt het argument record-nummer het aantal bytes aan dat naar het bestand moet worden geschreven. Het record-nummer mag dan niet groter zijn dan de lengte van de buffer die in de OPEN COM(n)-opdracht werd opgegeven.

XOR komt vooral van pas bij animaties. Als een afbeelding twee maal op een complexe achtergrond wordt weergegeven, wordt die achtergrond ongewijzigd opgeslagen. Op die wijze kan een object over het scherm worden verplaatst zonder daarbij de achtergrond te verstoren.

De standaard werkwijze is XOR.

Raadpleeg de beschrijving COLOR, PALETTE en SCREEN voor de effecten tijdens verschillende scherm-modi.

Animaties met objecten worden doorgaans als volgt uitgevoerd:

1 Stuur de objecten naar het beeldscherm.

2 Bereken de nieuwe positie van de objecten.

3 Plaats de objecten opnieuw op het beeldscherm, op de oude schermpositie, zodat de eerste afbeelding wordt overschreven.

4 Herhaal stap 1, maar plaats de objecten nu op de nieuwe positie.

Op deze wijze bewegen voorwerpen over het scherm zonder de achtergrond te wijzigen. Door de tijd voor terugkeer van stap 4 naar 1 zo kort mogelijk te houden en tussen stap 1 en 3 een langere tijdsduur toe te staan, wordt het flikkeren tot een minimum beperkt. Als meerdere objecten tegelijk worden gemanipuleerd, handel dan elk afzonderlijk object stap voor stap onmiddellijk af.

Als het intacthouden van de achtergrond niet belangrijk is, kunt u de animatie uitvoeren met behulp van PSET als argument.

Zorg dat rond de afbeelding een kader staat dat groter is dan de maximale afstand waarover de afbeelding wordt verplaatst. Dit kader zorgt dat bij het verplaatsen van objecten punten daadwerkelijk worden gewist. Deze werkwijze functioneert sneller dan de bovenomschreven procedure met XOR, omdat slechts één PUT-opdracht noodzakelijk is om het object te verplaatsen. Voorwaarde is wel dat de afbeelding die naar het bestand moet worden verplaatst groter is dan de bestaande afbeelding.

Voorbeeld:

```
10 CLS: SCREEN 1
20 PSET (130,120)
30 DRAW "U25;E7;R20;D32;L6;U12;L14"
40 DRAW "D12;L6": PSET(137,102)
50 DRAW "U4;E4;R8;D8;L12"
60 PSET (137,88)
70 DRAW "E4;R20;D32;G4":PAINT(131,119)
80 DIM A(500)
90 GET (125,130)-(170,80),A
100 FOR I=1 TO 1000:NEXT I
110 PUT (20,20),A,PSET
120 FOR I=1 TO 1000:NEXT I
130 GET (125,130)-(170,80),A
140 FOR I=1 TO 1000:NEXT I
150 PUT (220,130),A,PRESET
```

RANDOMIZE opdracht

Functie:

De toevalsgenerator (random number generator) voorzien van een nieuwe startwaarde, ook wel "seed" genoemd.

Syntaxis:

RANDOMIZE [*expressie*]
RANDOMIZE TIMER

Opmerkingen:

Als u geen *expressie* opgeeft, onderbreekt GW-BASIC de uitvoering van het programma en vraagt op de volgende wijze om een nieuwe startwaarde:

Onjuist willekeurig getal (-32768 to 32767)?

Als u de toevalsgenerator niet van een nieuwe startwaarde voorziet, zal het uitvoeren van de RND-opdracht steeds opnieuw dezelfde reeks toevalsgetallen genereren.
U kunt zorgen dat steeds een ander deel uit de reeks wordt gegenereerd door aan het begin van het programma bij een RANDOMIZE-opdracht een nieuwe startwaarde op te geven als het systeem daarom vraagt. Zie ook de beschrijving van de functie RND.

RANDOMIZE zonder toegevoegde *expressie* laat op het scherm de vraag om een nieuwe startwaarde verschijnen. Als u RANDOMIZE met een expressie gebruikt, worden decimale getallen niet afgerond tot een geheel getal. Voor *expressie* kunt u een willekeurige rekenkundige expressie opgeven.

U kunt ook een nieuwe startwaarde laten genereren zonder het systeem daarom te laten vragen, namelijk via de numerieke TIMER-functie:

RANDOMIZE TIMER

Voorbeeld 1:

```
10 RANDOMIZE TIMER
20 FOR I=1 TO 5
30 PRINT RND;
40 NEXT I
RUN
.88598 .484668 .586349 .119426 .709225
Ok

RUN
.803506 .162462 .929364 .292443 .322921
Ok
```

Bovenstaand voorbeeld maakt gebruik van het periodiek instellen van de interne klok.

Voorbeeld 2:

```
5 N=VAL (MID$(TIME$,7,2))  'Aantal seconden voor start-
  waarde
10 RANDOMIZE N             'Nieuwe startwaarde
20 PRINT N                 'Aantal seconden naar scherm
30 PRINT RND      'Gegenereerd toevalsgetal naar scherm
RUN
36
.2466638
Ok
RUN
37
.6530511
Ok
RUN
38
5.943847E-02
Ok
RUN
40
.8722131
Ok
```

De interne klok kan worden gebruikt voor het toekennen van een nieuwe startwaarde.

READ opdracht

Functie:
Gegevens uit DATA-opdrachten lezen en deze opslaan in variabelen.

Syntaxis:
READ *variabelenlijst*

Opmerkingen:
Een READ-opdracht functioneert alleen in combinatie met een of meer DATA-opdrachten.

READ slaat steeds één waarde uit een DATA-opdracht op in een variabele.

READ kan zowel tekenreeksen als numerieke gegevens lezen. Het type van de gegevens moet overeenkomen met het type van de variabele waarin zij moeten worden opgeslagen. Afwijken van deze regel leidt tot een syntaxis-fout.

Eén READ-opdracht kan gegevens lezen die over meerdere DATA-opdrachten zijn verdeeld. De gegevens worden altijd gelezen in de volgorde waarin ze in de DATA-opdrachten zijn opgenomen. Het is ook mogelijk meerdere READ-opdrachten gebruik te laten maken van dezelfde DATA-opdrachten.

Als het aantal variabelen in de *variabelenlijst* groter is dan het aantal gegevens in een of meer DATA-opdrachten dan volgt de foutmelding "Geen DATA meer".

Als het aantal variabelenamen in de variabelenlijst kleiner is dan het aantal gegevens in een of meer DATA-opdrachten, dan zal een volgende READ-opdracht doorgaan met lezen op de plaats waar de vorige READ-opdracht stopte. Als er geen nieuwe READ-opdracht volgt, worden de resterende gegevens genegeerd.

Met een RESTORE-opdracht kunt u een eerder gebruikte set gegevens opnieuw laten inlezen.

Voorbeeld 1:

```
.
.
.
80 FOR I=1 TO 10
90 READ A(I)
100 NEXT I
110 DATA 3.08,5.29,3.12,3.98,4.24
120 DATA 5.08,5.55,4.00,3.16,3.17
.
.
.
```

Via deze instructies worden de getallen uit de DATA-opdrachten gelezen en opgeslagen in cellen van matrix A. Na uitvoering van de instructies bevat cel A(1) de waarde 3.08, cel 2 de waarde 5.29 enz. De DATA-opdrachten (regels 110 en 120) mogen op een willekeurige plaats in het programma staan, zelfs voorafgaand aan READ.

Voorbeeld 2:

```
5 PRINT
10 PRINT "Naam", "Woonplaats", "Leeftijd"
20 READ N$,W$,L
30 DATA "Gosen,","Assen",37
40 PRINT N$, W$, L
RUN

Naam        Woonplaats      Leeftijd
Gosen,      Assen           37
Ok
```

Dit programma leest de tekenreeksen en numerieke gegevens uit de DATA-opdracht op regel 30 en geeft die op het scherm weer.

REM opdracht

Functie:

Het programma voorzien van opmerkingen voor de gebruiker.

Syntaxis:

REM[*opmerking*]
'[*opmerking*]

Opmerkingen:

REM-opdrachten worden tijdens de uitvoering van het programma genegeerd. Ze worden tijdens het "listen" letterlijk weergegeven.

Tijdens de uitvoering van een programma wordt alles wat op een regel volgt achter REM of een apostrof (') genegeerd. De uitvoering van het programma gaat door bij het eerstvolgende regelnummer.

Het is toegestaan bij GOTO en GOSUB te verwijzen naar regelnummers die beginnen met een REM-opdracht. De uitvoering van het programma gaat dan verder bij de eerstvolgende uitvoerbare opdracht. Verwijzen naar de eerste uitvoerbare opdracht komt de verwerkingssnelheid ten goede.

U kunt opmerkingen aan het einde van een programmaregel plaatsen. Tik direct vóór die opmerking een apostrof in plaats van het woord REM.

Attentie:

Gebruik in een DATA-opdracht nooit REM. Het programma beschouwt dit namelijk als tekst-tekenreeks en leest het woord dus als een gegeven.

Voorbeeld:

```
.
.
.
120 REM Getallen optellen
130 FOR I=1 TO 20
140 SOM=SOM + GETAL(I)
150 NEXT I
.
.
.
```

of:

```
.
.
.
120 FOR I=1 TO 20 'Getallen optellen
130 SOM=SOM + GETAL(I)
140 NEXT I
.
.
.
```

RENUM commando

Functie:
Wijzigen van regelnummers.

Syntaxis:
RENUM [*nieuw nummer*][,*oud nummer*][,*interval*]

Opmerkingen:
Met het argument [*nieuw nummer*] bepaalt u welk nummer als eerste nieuwe regelnummer zal worden toegekend. Het standaardnummer is 10.

Met [*oud nummer*] geeft u aan in welke programmaregel het hernummeren moet beginnen. Als u dit argument achterwege laat, begint het hernummeren in de eerste programmaregel.

Bij [*interval*] kunt u een waarde opgeven voor het interval tussen twee opeenvolgende regelnummers. De standaardwaarde is 10.

Bij het wijzigen van de regelnummers zorgt RENUM dat ook regelnummers achter ELSE, ERL, GOTO, GOSUB, THEN, ON...GOTO, ON...GOSUB, RESTORE en RESUME worden aangepast aan de nieuwe regelnummering. Als in een dergelijke opdracht wordt verwezen naar een niet bestaand regelnummer, dan verschijnt de foutmelding "Ongedefinieerd regelnummer x in y" (x is het niet bestaande regelnummer, y geeft de regel waar de foute verwijzing staat). RENUM zal het onjuiste regelnummer x niet aanpassen, maar verandert soms wel het regelnummer y.

Het is niet mogelijk met behulp van RENUM de volgorde van programmaregels te wijzigen. Dus RENUM 15,30 uitvoeren voor een programma met de drie regelnummers 10, 20 en 30 werkt niet. Als u tracht regelnummers met een waarde boven 65529 te gebruiken, volgt de foutmelding "Onjuiste functieaanroep".

Voorbeeld 1:
RENUM
Alle programmaregels krijgen een nieuw regelnummer. Het eerste nieuwe regelnummer is 10 en elk volgend regelnummer is steeds 10 groter.

Voorbeeld 2:
RENUM 300,,50
Alle programmaregels worden voorzien van een nieuw regelnummer. Het eerste nieuwe regelnummer is 300 en elk volgend regelnummer is steeds 50 groter.

Voorbeeld 3:

```
RENUM 1000,900,20
```

Alle programmaregels vanaf regel 900 worden opnieuw genummerd. Het eerste nieuwe regelnummer is 1000 en elk volgend regelnummer is steeds 20 groter.

RESET commando

Functie:

Alle diskette-bestanden sluiten en de directory-informatie opslaan op een diskette voordat die uit het station wordt verwijderd.

Syntaxis:

RESET

Opmerkingen:

Voer altijd een RESET-commando uit alvorens een diskette uit het station te nemen. Als u dat achterwege laat, zal de diskette bij later gebruik niet beschikken over een directory-spoor met de huidige gegevens over de directory.

RESET sluit alle geopende bestanden op alle diskettes en schrijft naar elke diskette met geopende bestanden een gewijzigde versie van de directory-informatie.

RESTORE commando

Functie:

Mogelijkheid bieden gegevens in DATA-opdrachten meermaals te laten lezen.

Syntaxis:

RESTORE [*regelnummer*]

Opmerkingen:

Als u een *regelnummer* opgeeft, leest de eerstvolgende READ-opdracht het eerste gegeven van de DATA-opdracht op de aangegeven regel.

Geeft u geen *regelnummer* op, dan leest READ het eerste gegeven in de eerstvolgende DATA-opdracht.

Voorbeeld:

```
10 READ A,B,C
20 RESTORE
30 READ D,E,F
40 DATA 57,68,79
```

De waarde 57 wordt zowel opgeslagen in variabele A als variabele D, 68 wordt opgeslagen in B en E enz.

RESUME opdracht

Functie:
Uitvoering van het programma hervatten nadat een routine voor foutafhandeling werd uitgevoerd.

Syntaxis:
RESUME
RESUME 0
RESUME NEXT
RESUME *regelnummer*

Opmerkingen:
De gekozen syntaxis bepaalt op welke plaats de uitvoering van een programma wordt hervat nadat een opgetreden fout werd afgehandeld:

syntaxis	effect
RESUME of RESUME 0	Uitvoering wordt hervat bij de opdracht waar de fout is opgetreden.
RESUME NEXT	Uitvoering wordt hervat bij de opdracht die direct volgt op de opdracht waar de fout is opgetreden.
RESUME *regelnummer*	Uitvoering wordt hervat bij het opgegeven regelnummer.

Als u een RESUME-opdracht gebruikt buiten een routine voor foutafhandeling, dan volgt de foutmelding "RESUME zonder fout".

Voorbeeld:

```
10 ON ERROR GOTO 900
.
.
900 IF (ERR=230) AND (ERL=90) THEN PRINT
    "Opnieuw!":RESUME 80
.
.
```

Als na uitvoering van regel 10 een fout optreedt, wordt de in regel 900 geprogrammeerde instructie uitgevoerd en vervolgens gaat de uitvoering van het programma verder op regel 80.

RETURN opdracht

Functie:
Terugkeren vanuit een subroutine naar het hoofdprogramma.

Syntaxis:
RETURN [*regelnummer*]

Opmerkingen:
Door de uitvoering van een RETURN-opdracht keert GW-BASIC terug naar de opdracht die direct volgt op de laatst uitgevoerde GOSUB-opdracht. In één subroutine mogen meerdere RETURN-opdrachten staan,

zodat GW-BASIC vanaf verschillende plaatsen in de routine naar het hoofdprogramma kan worden teruggestuurd. Subroutines mogen op een willekeurige plaats in het programma staan.

Door achter RETURN een regelnummer toe te voegen wordt de uitvoering van het programma hervat op de aangegeven regel. Deze optie wordt vooral gebruikt bij het detecteren van gebeurtenissen.

Als een bepaalde gebeurtenis wordt gedetecteerd, wordt automatisch een STOP-instructie uitgevoerd, zodat dezelfde gebeurtenis niet onmiddellijk opnieuw wordt gedetecteerd. RETURN in een detectie-routine voert automatisch ON uit, tenzij in de routine zelf een OFF-instructie wordt uitgevoerd.

Gebruik de constructie RETURN met regelnummer zeer zorgvuldig, want het systeem onthoudt GOSUB-, WHILE- en FOR-opdrachten die bij het aanroepen van de subroutine niet volledig waren afgehandeld, zodat fouten als "NEXT zonder FOR" kunnen optreden als u naar een onjuist regelnummer springt.

RIGHT$ functie

Functie:
Een tekenreeks vormen, bestaande uit het opgegeven aantal tekens vanaf het einde van een andere tekenreeks.

Syntaxis:
RIGHT$(*xS*,*n*)

Opmerkingen:
Als *n* gelijk is aan of groter is dan de lengte van *xS* (=LEN(xS)) dan wordt *xS* in zijn geheel opgeslagen in de nieuwe tekenreeks. Als *n* gelijk is aan 0, dan resulteert RIGHT$ in een nul-string (lengte 0, lege tekenreeks). Zie ook de beschrijving van LEFT$ en MID$.

Voorbeeld:
```
10 A$="Disk BASIC"
20 PRINT RIGHT$(A$,5)
RUN
BASIC
Ok
```

De laatste 5 tekens uit A$ worden op het scherm afgebeeld.

RMDIR functie

Functie:
Een subdirectory wissen.

Syntaxis:
RMDIR *padnaam*

Opmerkingen:
Argument *padnaam* bestaat uit een tekenreeks-expressie met een maximale lengte van 63 tekens, die aangeeft welke subdirectory uit welke hoofd-directory moet worden gewist.

De subdirectory die u wilt wissen, mag geen bestanden meer bevatten, behalve "." en "..". In andere gevallen treedt de fout "Pad/bestand niet toegankelijk" op.

Voorbeeld:

```
RMDIR "VERKOPEN\PETERS\RAPPORT"
```

Dit commando wist de subdirectory "RAPPORT".

RND functie

Functie:
Een toevalsgetal tussen 0 en 1 genereren.

Syntaxis:
RND [(*n*)]

Opmerkingen:
Bij elke aanroep van deze functie binnen een programma genereert de RND-functie dezelfde reeks toevalsgetallen. U kunt dit veranderen door de toevalsgenerator een nieuwe startwaarde te geven (ook wel "seed" genoemd) met behulp van de opdracht RANDOMIZE.

Als u voor *n* de waarde 0 opgeeft, wordt het laatst gegenereerde toevalsgetal herhaald. Als *n* groter is dan 0 of als u voor *n* geen waarde opgeeft, dan wordt het eerstvolgende toevalsgetal uit de reeks gegenereerd.

Onderstaande instructie genereert een toevalsgetal uit het bereik 0 t/m *n*, dat bovendien via INT wordt afgerond op een geheel getal:

```
INT(RND*(n+1))
```

U kunt de toevalsgenerator een nieuwe startwaarde toekennen door voor *n* een negatieve waarde in te tikken.

Voorbeeld:

```
10 FOR R=1 TO 5
20 PRINT INT(RND*101),
30 NEXT
RUN
53     30     31     5     76
Ok
```

Dit programma genereert vijf pseudo-random getallen uit het bereik van 0 tot 100.

RUN commando

Functie:
Het programma uitvoeren dat zich op dat moment in het geheugen bevindt, of een programma eerst van diskette laden en automatisch opstarten.

Syntaxis:
RUN [*regelnummer*][,r]
RUN *bestandsnaam*[,r]

Opmerkingen:

RUN, al dan niet gevolgd door een regelnummer, voert het programma uit dat zich in het werkgeheugen bevindt.

Als u een *regelnummer* opgeeft, begint de uitvoering van het programma in de aangegeven programmaregel. Geeft u geen *regelnummer* op, dan start de uitvoering in de eerste programmaregel.

Als u een RUN-commando laat uitvoeren terwijl er geen programma in het werkgeheugen is opgeslagen, dan keert GW-BASIC terug naar de commando-modus.

RUN gevolgd door een *bestandsnaam* sluit alle geopende bestanden en wist het programma dat zich op dat moment in het geheugen bevindt. Vervolgens wordt het opgegeven programma vanaf diskette in het geheugen geladen en automatisch opgestart.

Als u het argument [,r] toevoegt blijven alle geopende bestanden open.

Een RUN-commando breekt een op dat moment weergegeven geluid af en stelt Music Foreground (voorgrondmuziek) in. Bovendien worden automatisch de instructies STRIG OFF en PEN OFF uitgevoerd.

Voorbeeld:

```
RUN "STATIST",r
```

Dit commando voert het programma STATIST uit en laat alle vooraf al geopende bestanden open.

SAVE commando

Functie:

Een programma opslaan op diskette.

Syntaxis:

SAVE *bestandsnaam* [,a]
SAVE *bestandsnaam* [,p]

Opmerkingen:

De *bestandsnaam* moet tussen aanhalingstekens (") staan en moet voldoen aan de binnen MS-DOS geldende regels voor bestandsnamen. Als u de naam intikt van een reeds bestaande bestand, wordt dat bestand overschreven door de versie die u op dat moment opslaat. Geeft u bij de programmanaam geen bestandstype op, dan wordt automatisch het bestandstype .BAS voor een BASIC programma gekozen.

De optionele toevoeging a (door een komma gescheiden van de bestandsnaam) slaat het bestand als ASCII-bestand op. In andere gevallen slaat GW-BASIC gecomprimeerd op in binair format. ASCII-formaat vergt weliswaar meer opslagruimte op de diskette, maar voor sommige commando's (MERGE en MS-DOS opdrachten zoals TYPE) is het noodzakelijk dat een bestand in ASCII-formaat wordt opgeslagen.

De optionele toevoeging p (door een komma gescheiden van het bestandsnaam) dient als bescherming ("protection") voor het bestand. GW-BASIC slaat het bestand op in een gecodeerde binaire vorm. Een op deze wijze beschermde bestand kan later niet worden ge-list of gewijzigd. Het is daarbij wel verstandig een tweede versie van hetzelfde bestand zonder de optie p op te slaan, zodat u de mogelijkheid blijft behouden het programma later aan te passen.

Voorbeeld 1:

```
SAVE "COM2",a
```

Dit commando slaat het bestand COM2.BAS als ASCII-bestand op.

Voorbeeld 2:

```
SAVE "ELEK",p
```

Dit commando slaat het bestand ELEK.BAS op als beschermd binair bestand.

SCREEN functie

Functie:

ASCII-code bepalen voor het teken dat zich op de opgegeven schermpositie bevindt.

Syntaxis:

x=SCREEN(*rij,kolom*[,z])

Opmerkingen:

De ASCII-code (0-255) wordt opgeslagen in de variabele met de naam x.

Via het argument rij geeft u aan op welke rij (schermregel) het teken zich bevindt waarvan u de ASCII-code wilt weten. Daarvoor geeft u een getal op uit het bereik 1-40 of 1-80, afhankelijk van de op dat moment geactiveerde schermbreedte (regellengte). Zie de opdracht WIDTH.

Via het argument kolom geeft u aan op welke kolom (tekenpositie op de schermregel) het teken zich bevindt.

Argument z bestaat uit een geldige numerieke expressie met een waarde die waar (true) of onwaar (false) is. Dit argument functioneert alleen tijdens de alfanumerieke modus. In de numerieke expressie wordt het rangnummer van het opgevraagde teken opgeslagen. Als tijdens de alfanumerieke modus het optionele argument z wordt gebruikt en de waarde daarvan waar is (d.w.z. ongelijk aan 0), dan worden in plaats van de ASCII-code de kleurkenmerken van het teken gegeven. Zie ook de beschrijving van de opdracht COLOR.

Als voor een of meer argumenten een ongeldige waarde wordt opgegeven, treedt de fout "Onjuiste functieaanroep" op. U kunt voor rij alleen de waarde 25 opgeven als KEY ON (voor informatie over functietoetsen) niet is geactiveerd.

Voorbeeld 1:

```
100 X=SCREEN(10,10)
```

Als op de coördinaten 10,10 de letter A staat, wordt de daarbij behorende ASCII-code 65 opgeslagen in variabele X.

Voorbeeld 2:

```
100 X=SCREEN (1,1,1)
```

Deze instructie geeft de kleurkenmerken van het teken in de hoek linksboven in het scherm.

SCREEN opdracht

Functie:
Instellingen van het beeldscherm bepalen.

Syntaxis:
SCREEN [*modus*] [,[*kleur-optie*]] [,[*apag*]] [,[*zpag*]]

Opmerkingen:
De opdracht SCREEN wordt vooral gebruikt voor het kiezen van de scherm-modus die voor de gegèven combinatie van computer en monitor tot de beste resultaten leidt. Hieronder is aangegeven welke modi bij bepaalde systeemconfiguraties mogelijk zijn.

MDPA met monochroom monitor: modus 0
De IBM Monochrome Display and Printer Adapter (MDPA) is bestemd voor een monochroom beeldscherm en uitsluitend voor de tekst-modus.

CGA met kleurenmonitor: modi 0, 1 en 2
De Color Graphics Adapter (CGA), of een CGA-compatibele adapter, met een kleurenmonitor is een vaak gebruikte combinatie. Deze configuratie biedt mogelijkheden voor programma's in de tekst-modus, maar ook voor het gebruik van grafische patronen in medium en hoge resolutie.

EGA met kleurenmonitor: modi 0, 1, 2, 7 en 8
De vijf genoemde scherm-modi maken aansluiting van een kleurenmonitor mogelijk als het systeem over een Enhanced Graphics Adapter (EGA) of een EGA-compatibele adapter beschikt. Als de EGA-opties zijn ingesteld op CGA-compatibiliteit, dan draaien programma's tijdens de modi 1 en 2 op dezelfde wijze als onder CGA. De modi 7 en 8 komen overeen met de modi 1 en 2, met dit verschil dat in de modi 7 en 8 een uitgebreidere reeks kleuren ter beschikking staan.

EGA met Enhanced monitor: modi 0, 1, 2, 7 en 8
Bij de combinatie van EGA met een Enhanced kleurenmonitor, komen de vijf genoemde modi vrijwel overeen met de modi van de combinatie EGA met kleurenmonitor. Soms treden twee verschillen op:

1 Bij modus 0 kan de kleur van het kader niet dezelfde zijn als bij de configuratie EGA met kleurenmonitor, omdat op een Enhanced kleurenmonitor tijdens de tekst-modus met een schermresolutie van 640 x 350 het kader niet kan worden ingesteld .

2 Op het beeldscherm van een Enhanced kleurenmonitor is de kwaliteit van afgebeelde tekst beter, omdat de Enhanced monitor per teken 8 x 14 pixels gebruikt en de combinatie EGA met kleurenmonitor slechts 8 x 8 pixels.

EGA met Enhanced kleurenmonitor: modus 9
Deze modus benut alle capaciteiten van de Enhanced kleurenmonitor. Modus 9 maakt het mogelijk het hoogste oplossend vermogen te gebruiken dat door de configuratie EGA met Enhanced kleurenscherm kan worden gerealiseerd. Programma's die voor deze configuratie zijn geschreven, werken niet op andere systeemconfiguraties.

EGA met monochroom monitor: modus 10
De monochroom monitor is geschikt voor weergave van monochroom grafische patronen met een in modus 10 zeer hoge resolutie. Programma's die voor deze modus zijn ontwikkeld, werken niet op andere systeemconfiguraties.

Argumenten in de SCREEN-opdracht

Argument [*modus*]
Het argument [modus] bestaat uit een geheel getal, met de waarde 0, 1, 2, 7, 8, 9 of 10. Andere getallen zijn niet toegestaan. De keuze voor een bepaalde modus is, zoals uit de zojuist omschreven configuraties moge blijken, afhankelijk van de configuratie waarvoor het door u gebruikte programma is ontwikkeld.

Nu volgt een beschrijving van elke SCREEN-modus afzonderlijk:

SCREEN 0
- alleen tekst-modus
- 25 regels van 40 of 80 tekens tekst, waarbij elk teken is opgebouwd uit 8 x 8 beeldpunten (8 x 14 bij EGA)
- 2 kleurkenmerken, 16 kleuren per kenmerk
- met EGA: 16 kleurkenmerken, 16 kleuren per kenmerk

SCREEN 1
- 320 x 200 beeldpunten voor grafische weergave met medium resolutie
- 25 regels van elk 80 tekens tekst, tekens opgebouwd uit 8 x 8 pixels
- 4 kleurkenmerken, 16 kleuren per kenmerk
- zowel geschikt voor EGA als voor CGA
- 2 bits per pixel

SCREEN 2
- 640 x 200 beeldpunten voor grafische weergave met hoge resolutie
- 25 regels van elk 40 tekens tekst, tekens opgebouwd uit 8 x 8 pixels
- 2 kleurkenmerken, 16 kleuren per kenmerk
- zowel geschikt voor EGA als voor CGA
- 1 bit per pixel

SCREEN 7
- 320 x 200 beeldpunten voor grafische weergave met medium resolutie
- 25 regels van elk 40 tekens tekst, tekens opgebouwd uit 8 x 8 pixels
- 2, 4, of 8 geheugenpagina's bij resp. 64K, 128K of 256K geheugen op de EGA-kaart
- 16 kleurkenmerken, 16 kleuren per kenmerk
- alleen mogelijk met EGA
- 4 bits per pixel

SCREEN 8
- 640 x 200 beeldpunten voor grafische weergave met hoge resolutie
- 25 regels van elk 80 tekens tekst, tekens opgebouwd uit 8 x 8 pixels
- 1, 2, of 4 geheugenpagina's bij resp. 64K, 128K of 256K geheugen op de EGA-kaart
- 16 kleurkenmerken, 16 kleuren per kenmerk
- alleen mogelijk met EGA
- 4 bits per pixel

SCREEN 9
- 640 x 350 beeldpunten voor grafische weergave met Enhanced-resolutie
- 25 regels van elk 80 tekens tekst, tekens opgebouwd uit 8 x 14 pixels
- 16 kleurkenmerken, 64 kleuren per kenmerk (bij meer dan 64K EGA-geheugen) of 4 kleurkenmerken, 16 kleuren per kenmerk (bij 64K EGA-geheugen)
- pagina's met een lengte van 2 schermen bij 256K EGA-geheugen
- alleen mogelijk met EGA
- 2 bits per pixel (64K EGA-geheugen)
- 4 bits per pixel (meer dan 64K EGA-geheugen)

SCREEN 10
- 640 x 350 beeldpunten voor grafische weergave met Enhanced-resolutie
- 25 regels van elk 80 tekens tekst, tekens opgebouwd uit 8 x 14 pixels
- pagina's met een lengte van 2 schermen bij 256K EGA-geheugen
- 4 kleurkenmerken, 9 pseudo-kleuren
- alleen mogelijk met EGA
- 2 bits per pixel

Standaardkenmerken voor SCREEN 10 op een monochroom scherm:

Kenmerk-waarde	Afgebeelde pseudo-kleur
0	uit
1	aan, normale intensiteit
2	knipperend
3	aan, hoge intensiteit

Standaardkleuren voor SCREEN 10 op een monochroom scherm:

Kleur-waarde	Afgebeelde pseudo-kleur
0	uit
1	knipperend, van uit naar aan
2	knipperend, van uit naar hoge intensiteit
3	knipperend, van aan naar uit
4	aan
5	knipperend, van aan naar hoge intensiteit
6	knipperend, van hoge intensiteit naar uit
7	knipperend, van hoge intensiteit naar aan
8	hoge intensiteit

Argument [*kleur-optie*]
Zowel voor TV-ontvangers als voor monitoren met samengestelde video-signalen bestaat het argument [kleur-optie] uit een numerieke expressie die waar (true, ongelijk aan 0) of onwaar (false, gelijk aan 0) is. De waarde 0 schakelt de kleur uit, zodat alleen zwart/wit weergave mogelijk is. Is de waarde ongelijk aan 0, dan is kleurenweergave mogelijk. Bij SCREEN 0 is het effect van de waarde voor kleur-optie omgekeerd.

Argumenten [*apag*] en [*zpag*]
Deze argumenten zijn van belang indien uw configuratie over een EGA-kaart beschikt en voldoende geheugenruimte heeft voor het realiseren van pagina's die meerdere schermen lang zijn. Het argument [apag] bepaalt de "actieve" geheugenpagina en het argument [zpag] de "zichtbare" geheugenpagina. Het begrip actieve pagina duidt op dat deel van het geheugen waar grafische opdrachten zijn opgeslagen. De zichtbare pagina is het deel van het geheugen dat zichtbaar is op het beeldscherm.

Animaties kunnen worden gerealiseerd door afwisselend afbeelden van grafische pagina's. De opzet is de zichtbare pagina met voltooide grafische patronen af te beelden en intussen een of meer actieve pagina's met grafische opdrachten te laten uitvoeren. Een pagina wordt pas afgebeeld als de grafische uitvoer naar die pagina geheel is voltooid. Deze werkwijze komt tot uiting in instructies als:

```
SCREEN 7,,1,2 'Maak pagina 1 klaar en geef intussen pagina 2 weer
.
.  grafische uitvoer naar pagina 1
.  terwijl pagina 2 op het scherm blijft
.
SCREEN 7,,2,1 'Maak pagina 2 klaar en geef intussen pagina 1 weer
.
.  grafische uitvoer naar pagina 2
.  terwijl pagina 1 op het scherm blijft
.
```

Zoals uit onderstaande tabel blijkt, is het aantal beschikbare pagina's afhankelijk van de SCREEN-modus en de geheugencapaciteit:

Eigenschappen van de SCREEN-modi

Modus	Resolutie	Kenmerk Bereik	Kleuren-Bereik	EGA Geheugen	Pagina's	Pag.	Afm.
0	40 kol. tekst	n.v.t.	0-15[a]	n.v.t.		1	2K
	80 kol. tekst	n.v.t.	0-15[a]	n.v.t.		1	2K
1	320 x 200	0-3[b]	0-3	n.v.t.		1	16K
2	640 x 200	0-1[b]	0-1	n.v.t.		1	16K
7	320 x 200	0-15	0-15	64K		2	32K
				128K		4	
				256K		8	
8	640 x 200	0-15	0-15	64K		1	64K
				128K		2	
				256K		4	
9	640 x 350	0-3	0-15	64K		1	64K
		0-15	0-63	128K		1	128K
		0-15	0-63	256K		2	
10	640 x 350	0-3	0-8	128K		1	128K
				256K		2	

[a] Getallen uit het bereik 16-31 zijn knipperende versies van de kleuren 0-15
[b] Kenmerken alleen van toepassing bij EGA

Kenmerken en kleuren
De instelling van de kenmerken en kleuren variëren per scherm-modus en systeemconfiguratie. (Zie ook de beschrijving van de opdracht PALETTE voor een bespreking van de kenmerk- en kleurnummers.) De volgende tabel geeft een overzicht van de meeste kenmerk- en kleurconfiguraties.

Standaardkenmerken en kleuren

Kenmerk voor modi			kleurenmonitor		monochroom monitor	
1,9	2	0,7,8,9[b]	Nummer[c]	Kleur	Nummer[c]	Kleur
0	0	0	0	zwart	0	uit
		1	1	blauw		(onderstr)[a]
		2	2	groen	1	aan[a]
		3	3	cyaan	1	aan[a]
		4	4	rood	1	aan[a]
		5	5	magenta	1	aan[a]
		6	6	bruin	1	aan[a]
		7	7	wit	1	aan[a]
		8	8	grijs	0	uit
		9	9	lichtblauw		hoge int. (onderstr)
		10	10	lichtgroen	2	hoge int.
1		11	11	licht cyaan	2	hoge int.
		12	12	lichtrood	2	hoge int.
2		13	13	licht magenta	2	hoge int.
		14	14	geel	2	hoge int.
3	1	15	15	hoge int. wit	0	uit

a Uit indien gebruikt voor achtergrond
b Bij meer dan 64K EGA-geheugen
c Alleen voor monochroom in modus 0

De volgende tabel geeft de voorgrondkleuren voor de diverse modi.

Standaard voorgrondkenmerk		Standaard voorgrondkleur		
scherm-modus	Enhanced[a] kleur	mono-chroom	Enhanced[a] kleur	mono-chroom
0	7	7	7	1
1	3	n.v.t.	15	n.v.t.
2	1	n.v.t.	15	n.v.t.
7	15	n.v.t.	15	n.v.t.
8	15	n.v.t.	15	n.v.t.
9	3[b]	n.v.t.	63	n.v.t.
10	n.v.t.	3	n.v.t.	8

a Enhanced kleurenmonitor
b 15 bij meer dan 64K EGA-geheugen

SGN functie

Functie:
Het teken van een numerieke waarde aangeven.

Syntaxis:
SGN(x)

Opmerkingen:
Voor x kunt u een willekeurige numerieke expressie opgeven.

Als x positief is, is het resultaat van SGN(x) 1.
Als x 0 is, is het resultaat van SGN(x) ook 0.
Als x negatief is, is het resultaat van SGN(x) -1.

Deze opdracht lijkt enigszins op SIN(*x*), maar is niet hetzelfde. SIN(*x*) heeft betrekking op trigonometrische functies en radialen in plaats van met de getallen 0 en 1 te werken.

Voorbeeld:

```
10 INPUT "Tik een getal in",X
20 ON SGN(X)+2 GOTO 100, 200, 300
```

Als X negatief is, springt het programma naar regel 100, bij X=0 naar regel 200 en als X groter dan 0 is naar regel 300.

SHELL opdracht

Functie:

Een ander programma of een ander batch-bestand laden en uitvoeren. Daarna terugkeren naar het GW-BASIC programma en de uitvoering hervatten bij de opdracht die direct volgt op de SHELL-opdracht. Een programma dat onder besturing van GW-BASIC wordt uitgevoerd, wordt een "child process" genoemd.

Syntaxis:

SHELL [*tekenreeks*]

Opmerkingen:

Het optionele argument *tekenreeks* bestaat uit een geldige tekenreeks-expressie met de naam van het uit te voeren programma en (optionele) commando-argumenten. De programmanaam in het argument *tekenreeks* mag elk binnen MS-DOS COMMAND.COM geldig bestandstype hebben. Als de naam geen bestandstype bevat, zoekt COMMAND naar een bestand met het bestandstype .COM, vervolgens naar het bestandstype .EXE en tenslotte naar bestanden met het bestandstype .BAT. Als geen van deze typen bestanden wordt gevonden, geeft SHELL de foutmelding "Bestand niet gevonden".

Tekst die van de programmanaam wordt gescheiden door een lege tekenpositie wordt door COMMAND beschouwd als programmaparameter.

Tijdens de uitvoering van het "child process" blijft GW-BASIC in het geheugen opgeslagen. Zodra dit proces is voltooid, wordt de uitvoering van het BASIC programma hervat met de opdracht die direct volgt op de SHELL-opdracht.

SHELL zonder toegevoegde tekenreeks draagt de besturing over aan MS-DOS, waarna u elke instructie mag geven die tijdens MS-DOS is toegestaan. Met behulp van het MS-DOS commando EXIT kunt u terugkeren naar GW-BASIC.

Voorbeeld:

```
SHELL
A>DIR
A>EXIT
Ok
10 OPEN "SORTIN.DAT" FOR OUTPUT AS #1
20 'Sla een hoeveelheid gegevens op om
21 'die te laten sorteren
.
.
.
1000 CLOSE 1
1010 SHELL "SORT <SORTIN.DAT   >SORTOUT.DAT"
1020 OPEN "SORTOUT.DAT" FOR INPUT AS #1
1030 'Verwerk de gesorteerde gegevens
```

Laat een hoeveelheid gegevens sorteren met behulp van SHELL SORT en lees die gegevens vervolgens in om ze te verwerken in een overzicht.

SIN functie

Functie:

Berekening van de sinus voor een hoek in radialen.

Syntaxis:

SIN(*x*)

Opmerkingen:

De waarde van *x* is een hoek in radialen. De sinus wordt berekend met enkele precisie, tenzij bij het opstarten van GW-BASIC via de optie /d de dubbele precisie is geactiveerd.

Om de sinus te laten berekenen voor een hoek in graden, wijzigt u de instructie in SIN(x*PI/180).

Voorbeeld:

```
PRINT SIN(1.5)
.9974951
Ok
```

De sinus voor een hoek van 1,5 radialen is .9974951 (berekend met enkele precisie).

SOUND opdracht

Functie:

Geluiden weergeven via de luidspreker.

Syntaxis:

SOUND *frequentie,tijdsduur*

Opmerkingen:

Met het argument *frequentie* bepaalt u de frequentie van de opgewekte toon in Hertz (trillingen per seconde). *Frequentie* moet een getal zijn uit het bereik 37 tot 32767.

Het tweede argument bepaalt de tijdsduur van de toon. De interne klok tikt 18,2 maal per seconde. De *tijdsduur* moet een getal zijn uit het bereik 0-65535. Om een tijdsduur van t seconden op te geven, tikt u een waarde van t maal 18,2.

Als u voor het argument *tijdsduur* de waarde 0 gebruikt, breekt u de geluidsweergave af. Als op dat moment geen geluid wordt weergegeven, heeft de waarde 0 uiteraard geen effect.

De status van de opdracht PLAY bepaalt of het geluid op de voorgrond of achtergrond wordt verwerkt.

Voorbeeld:

```
2500 SOUND RND*1000+37,2
2600 GOTO 2500
```

Deze twee instructies produceren kortdurende willekeurige geluiden.

Onderstaande tabel geeft een overzicht van het verband tussen noten en frequenties in de twee octaven rond midden C.

Verband tussen noten en frequenties

Noot	Frequentie	Noot	Frequentie
C	130.810	C*	523.250
D	146.830	D	587.330
E	164.810	E	659.260
F	174.610	F	698.460
G	196.000	G	783.990
A	220.000	A	880.000
B	246.940	B	987.770
C	261.630	C	1046.500
D	293.660	D	1174.700
E	329.630	E	1318.500
F	349.230	F	1396.900
G	392.000	G	1568.000
A	440.000	A	1760.000
B	493.880	B	1975.500

* midden C

Door de frequentie te verdubbelen of te halveren kunt u de corresponderende noten uit het volgende, resp. vorige octaaf berekenen.

Als u een stilte wilt inlassen, gebruik dan de volgende instructie:

SOUND 32767, *tijdsduur*

Om de tijdsduur van een teleenheid te berekenen, deelt u het aantal teleenheden per minuut door het aantal tikken van de klok per minuut (1092).

De volgende tabel geeft het verband tussen teleenheden en aantal tikken van de interne klok

Verband tussen teleenheden en klok-tikken

tempo	benaming	teleenheden/ minuut	tikken/ teleenheid
zeer langzaam	largo assai		
	largo	40-66	27.3-18.2
	larghetto	60-66	18.2-16.55
	grave		
	lento		
	adagio	66-76	16.55-14.37
langzaam	adagietto		
	andante	76-108	14.37-10.11
gemiddeld	andantino		
	moderato	108-120	10.11-9.1
snel	allegretto		
	allegro	120-168	9.1-6.5
	vivace		
	veloce		
	presto	168-208	6.5-5.25
zeer snel	prestissimo		

SPACE$ functie

Functie:
Een tekenreeks maken die bestaat uit een bepaald aantal spaties.

Syntaxis:
SPACE$(*x*)

Opmerkingen:
De waarde van *x* wordt afgerond tot een geheel getal en moet tussen 0 en 255 liggen (zie ook de functie SPC).

Voorbeeld:

```
10 FOR N=1 TO 5
20 X$=SPACE$(N)
30 PRINT X$;N
40 NEXT N
RUN
 1
  2
   3
    4
     5
Ok
```

Regel 20 voegt steeds een spatie toe.

SPC functie

Functie:
Aantal spaties invoegen tijdens het uitvoeren van de opdrachten PRINT of LPRINT.

Syntaxis:
SPC(*n*)

Opmerkingen:
De waarde van *n* moet tussen 0 en 255 liggen. Als *n* groter is dan de op dat moment geactiveerde schermbreedte dan wordt de waarde van *n* automatisch ingesteld op "n MOD schermbreedte"

GW-BASIC neemt aan dat na SPC(*n*) een puntkomma is ingetikt, zodat eropvolgende tekens op dezelfde regel worden afgebeeld of afgedrukt.

SPC kan alleen worden toegepast binnen de opdrachten PRINT, LPRINT en PRINT# (zie ook de functie SPACE$).

Voorbeeld:
```
PRINT "hier" SPC(10) "en" SPC(10) "daar"
hier          en          daar
Ok
```

SQR functie

Functie:
Vierkantswortel berekenen.

Syntaxis:
SQR(*x*)

Opmerkingen:
Voor *x* mag nooit een negatief getal worden opgegeven.

De wortel uit *x* wordt met enkele precisie berekend, behalve wanneer bij het opstarten van GW-BASIC met behulp van de optie /d de dubbele precisie is geactiveerd.

Voorbeeld:
```
10 FOR X=10 TO 25 STEP 5
20 PRINT X; SQR(X)
30 NEXT
RUN
10  3.162278
15  3.872984
20  4.472136
25  5
Ok
```

STICK functie

Functie:
De x- en y-coördinaten van een joystick aangeven.

Syntaxis:
c=STICK(*n*)

Opmerkingen:

Het resultaat van de functie STICK wordt opgeslagen in variabele c.

De waarde van argument *n* moet 0, 1, 2 of 3 zijn.

waarde van n	resultaat
0	x-coördinaat van joystick A. Slaat de x- en y-coördinaten op van beide joysticks voor de volgende drie functie-aanroepen.
1	y-coördinaat van joystick A.
2	x-coördinaat van joystick B.
3	y-coördinaat van joystick B.

STOP opdracht

Functie:

Uitvoering van het programma beëindigen en overgaan naar commando-modus.

Syntaxis:

STOP

Opmerkingen:

STOP-opdrachten kunnen op willekeurige plaatsen in het programma staan. Na uitvoering van STOP verschijnt de mededeling:

`Onderbroken in regel` *nnnn*

waarbij *nnnn* aangeeft in welke programmaregel het programma werd gestopt.

Door uitvoering van STOP worden geen geopende bestanden gesloten, zoals bij END wel het geval is.

Na een STOP-opdracht keert GW-BASIC altijd terug naar de commando-modus. U kunt de uitvoering van het programma dan hervatten door het commando CONT in te tikken.

Voorbeeld:

```
10 INPUT A,B,C
20 K=A^2*5.3: L=B^3/.26
30 STOP
40 M=C*K+100:PRINT M
RUN
? 1,2,3
Break in line 30
Ok
PRINT L
30.76923
Ok
CONT
115.9
Ok
```

STR$ functie

Functie:
De waarde van een getal omzetten in een tekenreeks.

Syntaxis:
STR$(*x*)

Opmerkingen:
STR$(*x*) voert de omgekeerde bewerking uit van VAL(x$) (zie de functie VAL).

Voorbeeld:

```
10 INPUT "Tik een getal in: ",G
20 ON LEN(STR$(G)) GOSUB 30,40,50
.
.
.
```

De lengte van het getal bepaalt naar welke subroutine het programma springt.

STRIG opdracht en functie

Functie:
Status van de vuurknoppen van de joysticks aangeven.

Syntaxis als opdracht:
STRIG ON
STRIG OFF

Syntaxis als functie:
x=**STRIG**(*n*)

Opmerkingen:
Het resultaat wordt opgeslagen in de numerieke variabele *x*.

Argument *n* moet een geheel getal zijn uit het bereik 0 t/m 7.

De functie STRIG(*n*) kan alleen worden uitgevoerd na een STRIG ON-instructie. Na uitvoering van STRIG ON controleert GW-BASIC steeds of een vuurknop wordt ingedrukt. Met STRIG OFF schakelt u die detectie weer uit.

Onderstaande tabel geeft het effect voor elke waarde van *n*:

n	resultaat
0	-1 als vuurknop A1 werd ingedrukt sinds de laatste STRIG(0) opdracht en 0 als die vuurknop niet meer werd ingedrukt.
1	-1 als vuurknop A1 op dat moment wordt ingedrukt en 0 als dat niet zo is.
2	-1 als vuurknop B1 werd ingedrukt sinds de laatste STRIG(2) opdracht en 0 als die vuurknop niet meer werd ingedrukt.
3	-1 als vuurknop B1 op dat moment wordt ingedrukt en 0 als dat niet zo is.
4	-1 als vuurknop A2 werd ingedrukt sinds de laatste STRIG(4) opdracht en 0 als die vuurknop niet meer werd ingedrukt
5	-1 als vuurknop A2 op dat moment wordt ingedrukt en 0 als dat niet zo is.
6	-1 als vuurknop B2 werd ingedrukt sinds de laatste STRIG(6) opdracht en 0 als die vuurknop niet meer werd ingedrukt.
7	-1 als vuurknop B2 op dat moment wordt ingedrukt en 0 als dat niet zo is.

STRIG(n) opdracht

Functie:

Gebruik van joysticks mogelijk maken door de detectie van de vuurknoppen in- of uit te schakelen.

Syntaxis:

STRIG(*n*) ON
STRIG(*n*) OFF
STRIG(*n*) STOP

Opmerkingen:

Argument *n* kan een van de waarden 0, 2, 4 of 6 hebben, corresponderend met de vuurknoppen op beide joysticks, waarbij geldt:

0 voor knop A1
2 voor knop B1
4 voor knop A2
6 voor knop B2

`STRIG(n) ON`

Schakelt de detectie in, waarna GW-BASIC steeds controleert of de betreffende knop is ingedrukt.

`STRIG(n) OFF`

Schakelt de detectie uit, zodat GW-BASIC niet langer controleert of de vuurknop is ingedrukt.

`STRIG(n) STOP`

GW-BASIC reageert niet onmiddellijk op het indrukken van de betreffende vuurknop, maar onthoudt wel dat die werd ingedrukt. Na de eerstvolgende STRIG ON-opdracht wordt alsnog gereageerd op het indrukken van de vuurknop.

STRING$ functie

Functie:
Een tekenreeks ("string") met een bepaalde lengte samenstellen, bestaande uit het teken waarvan de ASCII-code wordt opgegeven. Een tweede mogelijkheid is het eerste teken van een opgegeven tekenreeks geven.

Syntaxis:
STRING$(*n*,*j*)
STRING$(*n*,*xS*)

Opmerkingen:
STRING$ is ook zeer nuttig voor het samenstellen van kaders of kop- en voetteksten op het scherm of via de printer.

Argument *n* bepaalt de lengte van de tekenreeks. Argument *j* is de ASCII-code van het gewenste teken. Voor beide argumenten moet een geheel getal tussen 0 en 255 worden opgegeven.

Met behulp van de tweede syntax-vorm krijgt u het eerste teken uit *xS*.

Voorbeeld:
```
10 X$ = STRING$(10,45)
20 PRINT X$"Maandoverzicht" X$
RUN
----------Maandoverzicht----------
Ok
```

De waarde 10 bepaalt hoe vaak het teken met ASCII-code 45 (het minteken) wordt afgedrukt.

Appendix C geeft een overzicht van de ASCII-codes.

SWAP opdracht

Functie:
De inhoud van twee variabelen verwisselen.

Syntaxis:
SWAP *variabele1,variabele2*

Opmerkingen:
Van elk type variabele (gehele getallen, enkele of dubbele precisie, tekenreeks) kan de inhoud worden uitgewisseld met een andere variabele. Bij een dergelijke uitwisseling tussen twee variabelen geldt echter altijd de voorwaarde dat beide variabelen van hetzelfde type moeten zijn. Wanneer van deze regel wordt afgeweken, treedt de fout "Onjuist type" op.

Voorbeeld:

```
10 A$="alles ":B$="niets ":C$="of "
20 PRINT A$ C$ B$
30 SWAP A$,B$
40 PRINT A$ C$ B$
RUN
alles of niets
niets of alles
Ok
```

Regel 30 verwisselt de inhoud van de twee tekenreeksen A$ en B$.

SYSTEM commando

Functie:
Systeembesturing overdragen aan MS-DOS.

Syntaxis:
SYSTEM

Opmerkingen:
Als u uw BASIC programma wilt bewaren, sla het dan eerst op en geef daarna pas het SYSTEM commando, anders gaat het programma verloren.

Het commando SYSTEM sluit alle bestanden en draagt vervolgens de besturing over aan MS-DOS. Als u GW-BASIC via een batch-bestand vanuit MS-DOS hebt aangeroepen, keert u na het SYSTEM-commando terug naar dat batch-bestand. Deze wordt vervolgens verder uitgevoerd, te beginnen op het punt waar eerder GW-BASIC werd aangeroepen.

Voorbeeld:

```
SYSTEM
A>
```

TAB functie

Functie:
Spaties invoegen tot een aangegeven regelpositie.

Syntaxis:
TAB(*n*)

Opmerkingen:
Als de cursor al op een positie rechts van de bij argument *n* opgegeven positie staat, dan zal de cursor na een TAB-instructie naar positie *n* op de eerstvolgende regel gaan.

Spatie 1 is de meest linkse regelpositie. De meest rechtse positie is de ingestelde schermbreedte. Voor argument *n* moet een waarde tussen 1 en 255 worden opgegeven.

Als de TAB-functie als laatste gegeven in een DATA-regel staat, springt de cursor niet naar de volgende regel. Het effect komt in dit geval overeen met het effect van een puntkomma.

TAB kan alleen worden gebruikt als onderdeel van de opdrachten PRINT, LPRINT of PRINT#. Raadpleeg ook de beschrijving van SPC.

Voorbeeld:

```
10 PRINT "Naam" TAB(35) "Bedrag": PRINT
20 READ A$,B$
30 PRINT A$ TAB(35) B$
40 DATA "Helga v.d. Meer","f 72,-"
RUN
Naam                    Bedrag
Helga v.d. Meer         f 72,-
Ok
```

TAN functie

Functie:
Berekening van de tangens voor een hoek in radialen.

Syntaxis:
TAN(*x*)

Opmerkingen:
De expressie *x* kan elk type numerieke waarde zijn. De berekening wordt in enkele precisie uitgevoerd, behalve wanneer via de optie /d de dubbele precisie is geactiveerd.

Als de waarden te groot worden, treedt de fout "Te groot getal" op en verschijnt het grootste getal dat GW-BASIC kan weergeven, inclusief het bijbehorende teken. De uitvoering van het programma gaat vervolgens door.

Als u de tangens wilt berekenen van een hoek die in graden wordt opgegeven, gebruikt u de instructie TAN(x*PI/180).

Voorbeeld:

```
10 Y=TAN(2)
20 PRINT "Tangens v.e. hoek van 2 radialen is "y
RUN
Tangens v.e. hoek van 2 radialen is -2.18504
Ok
```

TIME$ opdracht en variabele

Functie:
Instellen of weergeven van de juiste tijd.

Syntaxis als opdracht:
TIME$=*tekenreeks-expressie*

Syntaxis als variabele:
tekenreeks-expressie=TIME$

Opmerkingen:
Voor het argument *tekenreeks-expressie* geeft u een variabele op of een tekenreeks tussen aanhalingstekens. Met behulp van de tekenreeks-expressie kunt u instellen: uren (hh), uren en minuten (hh:mm) of uren, minuten en seconden (hh:mm:ss).

Met hh stelt u uren (0-23) in, waarbij voor de minuten en seconden de standaardwaarde 00 wordt gehanteerd.

Met hh:mm stelt u uren en minuten (0-59) in. Seconden wordt ingesteld op de standaardwaarde 00.

Met hh:mm:ss stelt u uren, minuten en seconden (0-59) in.

Als de *tekenreeks-expressie* geen geldige tekenreeks is, volgt de foutmelding "Onjuist type".

Als u een van de bovengenoemde waarden instelt, mag u in voorkomende gevallen een 0 aan het begin weglaten, maar er moet voor de eerste dubbele punt altijd minimaal één cijfer staan. U mag dus half 1 in de nacht niet intikken als TIME$=":30", maar wel als TIME$="0:30".

Als u een waarde opgeeft buiten het geldige bereik, dan treedt de fout "Onjuiste functieaanroep" op en wordt de eerder ingestelde tijd niet gewijzigd.

Door de waarde van TIME$ toe te kennen aan een variabele, slaat u de juiste tijd in die variabele op.

Als u TIME$ gebruikt in een LET- of PRINT-opdracht, dan wordt de juiste tijd afgebeeld en opgeslagen in de *tekenreeks-variabele.*

Bij de vorm *tekenreeks-expressie* = TIME$ geeft TIME$ een tekenreeks van 8 tekens in de vorm hh:mm:ss.

Voorbeeld 1:

```
TIME$ = "08:00"
Ok
PRINT TIME$
08:00:05
Ok
```

Bovenstaande instructie stelt de tijd in op 8:00 uur 's ochtends.

Voorbeeld 2:

```
10 KEY OFF: SCREEN 0: WIDTH 80: CLS
20 LOCATE 25,5
30 PRINT DATE$,TIME$
40 SEC=VAL(MID$(TIME$,7,2))
50 IF SEC=SSEC THEN 20 ELSE SSEC=SEC
60 IF SEC=0 THEN 1010
70 IF SEC=30 THEN 1020
80 IF SEC<57 THEN 20
.
.
.
1000 SOUND 1000,2:GOTO 20
1010 SOUND 2000,8:GOTO 20
1020 SOUND 400,4:GOTO 20
```

Het bovenstaande programma schrijft de datum en de tijd op regel 25 van het beeldscherm en geeft om de halve en om de hele minuut een signaal.

TIMER functie

Functie:

Aangeven hoeveel seconden zijn verlopen sinds middernacht of sinds de laatste reset van het systeem. Dit aantal seconden wordt berekend met enkele precisie.

Syntaxis:
v=TIMER

Opmerkingen:
Het aantal seconden wordt afgerond op de dichtstbijliggende gehele seconde. De waarde van de functie TIMER kan alleen worden gelezen, niet veranderd.

TRON en TROFF commando's

Functie:
Het verloop van de uitvoering van het programma continu volgen.

Syntaxis:
TRON
TROFF

Opmerkingen:
TRON (trace on) is een zeer geschikt hulpmiddel bij het foutzoeken in programma's. Als TRON is geactiveerd, verschijnt het regelnummer van elke programmaregel op het moment dat de betreffende regel wordt uitgevoerd. Het regelnummer staat tussen vierkante haken.

U kunt TRON zowel in de commando-modus als in de programma-modus gebruiken.

Met behulp van het commando TROFF (trace off) schakelt u TRON uit. Het commando NEW schakelt automatisch TRON uit.

Voorbeeld:
```
TRON
Ok
10 K=10
20 FOR J=1 TO 2
30 L=K+10
40 PRINT J;K;L
50 K=K+10
60 NEXT
70 END
RUN
[10][20][30][40] 1   10   20
[50][60][30][40] 2   20   30
[50][60][70]
Ok
TROFF
Ok
```

UNLOCK opdracht

Functie:
Bestanden die eerder via LOCK ontoegankelijk werden gemaakt weer toegankelijk maken. Deze opdracht wordt gebruikt in netwerken.

Syntaxis:
UNLOCK [#]*n*[,[*record-nummer*][TO *record-nummer*]]

Opmerkingen:

Voor *n* geeft u het nummer op dat in het programma aan dat bestand werd toegekend.

Het argument *record-nummer* verwijst naar het record waarvoor de via LOCK aangebrachte afsluiting moet worden opgeheven. Als u de afsluiting voor een reeks records wilt opheffen, geeft u in het eerste *record-nummer* het begin van de reeks op en achter **TO** het einde.

Voor record-nummers moet een waarde worden opgegeven van minimaal 1 en maximaal 2^{32}-1. De maximale afmeting van een record is 32767 bytes.

Bij het opgeven van een reeks records moet altijd eerst het laagste en als tweede het hoogste *record-nummer* worden opgegeven. Als u geen *record-nummer* opgeeft voor het begin van de record-reeks, dan kiest het systeem daarvoor automatisch de waarde 1.
Geeft u geen *record-nummer* op voor het einde van de reeks, dan wordt alleen voor het als eerste opgegeven record de afsluiting opgeheven.

Enkele voorbeelden van geldige UNLOCK opdrachten:

UNLOCK #n	heft de afsluiting op voor het gehele bestand n
UNLOCK #n,X	heft de afsluiting op voor record X
UNLOCK #n,TO Y	heft de afsluiting op voor de records 1 t/m Y
UNLOCK #n,X TO Y	heft de afsluiting op voor de records X t/m Y

De afsluiting voor een bestand of record moet worden opgeheven voordat het bestand of het record worden gesloten. Als u een UNLOCK-opdracht achterwege laat, bestaat de kans dat een bestand of record in een netwerk later niet meer toegankelijk is.

Als een UNLOCK-opdracht met juiste syntaxis om andere redenen niet kan worden uitgevoerd, dan verschijnt de melding "Schijf heeft schrijfbeveiliging". Elke LOCK-opdracht moet corresponderen met een UNLOCK-opdracht die naar dezelfde bestanden of records verwijst. Dit leidt vaak tot kort op elkaar volgende paren LOCK/UNLOCK-opdrachten.

Een voorbeeld van een correcte combinatie van LOCK- en UNLOCK-opdrachten:

```
LOCK #1, 1 TO 4
LOCK #1, 5 TO 8
UNLOCK #1, 1 TO 4
UNLOCK #1, 5 TO 8
```

Een voorbeeld van foute combinatie van LOCK en UNLOCK:

```
LOCK #1, 1 TO 4
LOCK #1, 5 TO 8
UNLOCK #1, 1 TO 8
```

USR functie

Functie:
Een assembleerroutine aanroepen

Syntaxis:
v=USR[*n*](*argument*)

Opmerkingen:
Met het optionele argument *n* bepaalt u welke USR-routine wordt aangeroepen. U mag waarden opgeven uit het bereik 0-9. Als u geen waarde voor *n* opgeeft, kiest het systeem automatisch USR0. Raadpleeg de beschrijving van DEF USR voor meer bijzonderheden over argument *n*.

Voor *argument* mag elke willekeurige numerieke of tekenreeks-expressie worden opgegeven.

Hoewel de CALL-opdracht het meest geschikte is om assembleerroutines aan te roepen, kunt u daarvoor ook de USR-functie gebruiken. Raadpleeg appendix D voor een vergelijking tussen CALL en USR en voor bijzonderheden over het aanroepen van assembleerroutines.

Als u een ander segment gebruikt dan het standaard data-segment (DS), dan moet een USR-instructie worden voorafgegaan door een DEF SEG-opdracht. Dit garandeert dat het code-segment naar de aangeroepen routine verwijst.

Het segment-adres in de DEF SEG-opdracht bepaalt het beginsegment van de subroutine.

Elke USR-functie moet corresponderen met een DEF USR-opdracht, waarmee vooraf de startwaarde voor het aanroepen is opgegeven. Die startwaarde en het op dat moment geactiveerde segment-adres uit de DEF SEG-opdracht bepalen het startadres van de aangeroepen subroutine.

Als in een programma meer dan 10 USR-routines nodig zijn, kan het aantal DEF USR[*n*]-opdrachten worden gebruikt dat noodzakelijk is om het startadres USR[*n*] voor elke routine opnieuw te definiëren.

Het type van de variabele *v*, waarin de waarde van USR wordt opgeslagen, moet overeenkomen met het doorgegeven argument. Als voor de betreffende assembleerroutine geen argument behoeft te worden doorgegeven, moet een zogenaamd "dummy-argument" worden ingetikt.

VAL functie

Functie:
De inhoud van een tekenreeks omzetten in een numerieke waarde.

Syntaxis:
VAL(x$)

Opmerkingen:
De functie VAL negeert lege tekenposities, spaties en line feeds die aan het begin van x$ staan. Stel dat u de volgende instructie intikt:

```
PRINT VAL ("  -3")
```

Dit leidt tot de numerieke waarde -3.

Als het eerste teken in x$ geen cijfer is, geeft VAL(x$) de waarde 0.

STR$ voert de omgekeerde bewerking uit van VAL, namelijk numerieke waarden omzetten in tekenreeksen.

Voorbeeld:

```
10 READ NAAM$,PLTS$,PROV$,JAAR$
20 IF VAL(JAAR$)<20 OR VAL(JAAR$)>75 THEN PRINT NAAM$ TAB(25)"Buiten gewenste bereik"
30 IF VAL(JAAR$)>=20 OR VAL(JAAR$)<=75 THEN PRINT NAAM$ TAB(25)"Accoord; tussen 20 en 75 jr."
.
.
.
```

VARPTR functie

Functie:

Aangeven op welk geheugenadres een variabele of het File Control Block (FCB) voor een bestand is opgeslagen.

Syntaxis:

VARPTR(*variabelenaam*)
VARPTR(*#bestandsnummer*)

Opmerkingen:

VARPTR wordt vaak gebruikt om na te gaan op welk geheugenadres een variabele of een matrix is opgeslagen, om dat adres te gebruiken wanneer de waarde van die variabele of die matrix moet worden doorgegeven aan een subroutine. Voor het opzoeken van het adres van een matrix kunt u de volgende instructie gebruiken:

```
VARPTR (A(0))
```

Vraag alleen het adres van een matrix op als u alle andere typen variabelen al hebt gedeclareerd. De geheugenadressen van matrices veranderen namelijk bij elke toevoeging van nog niet eerder gebruikte variabelen.

VARPTR (*#bestandsnummer*) geeft het startadres van een GW-BASIC File Control Block dat aan het betreffende *bestandsnummer* is toegekend.

VARPTR (*variabelenaam*) geeft het geheugenadres van de eerste byte met gegevens die in die variabele zijn opgeslagen. In de variabele met de opgegeven naam moet een waarde zijn opgeslagen voordat u VARPTR gebruikt, anders volgt de foutmelding "Onjuiste functieaanroep". Voor variabelenaam mag elk variabele-type worden gebruikt (numeriek, tekenreeks of matrix).

Het door VARPTR aangegeven adres is een geheel getal uit het bereik -32768 tot 32767. Als VARPTR een negatief adres geeft, moet daarbij 65535 worden opgeteld om het werkelijke geheugenadres te verkrijgen.

De volgende tabel geeft een overzicht van de startadressen voor informatie in het File Control Block van het door VARPTR aangegeven adres:

Startadressen van FCB-informatie

begin-adres	lengte	naam	omschrijving
0	1	modus	Modus waarvoor het bestand werd geopend: 1 alleen Input 2 alleen Output 4 Random I/O 16 alleen Append 32 intern gebruik 64 nog niet in gebruik 128 intern gebruik
1	38	FCB	diskette File Control Block
39	2	CURLOC	Aantal gelezen of geschreven sequentieel toegankelijke sectoren. Bij onbepaald benaderbare bestanden: laatst gelezen of geschreven record-nummer +1.
41	1	ORNOFS	Aantal bytes in de sector tijdens lezen of schrijven.
42	1	NMLOFS	Aantal bytes in INPUT-buffer.
43	3	***	Nog niet in gebruik.
46	1	DEVICE	Device-nummer: 0-9 stations A: t/m J: 255 KYBD: (toetsenbord) 254 SCRN: (beeldscherm) 253 LPT1: (printer 1) 252 CAS1: (cassette 1) 251 COM1: (communicatiepoort 1) 250 COM2: (communicatiepoort 2) 249 LPT2: (printer 2) 248 LPT3: (printer 3)
47	1	WIDTH	Device-breedte.
48	1	POS	Positie in buffer voor PRINT.
49	1	FLAGS	Voor interne doeleinden bij BLOAD en BSAVE. Niet voor gegevensbestanden.
50	1	OUTPOS	Uitvoerpositie tijdens tab-uitbreiding.
51	128	BUFFER	Fysieke gegevensbuffer voor uitwisseling van gegevens tussen DOS en BASIC. Gebruik dit adres voor controle van gegevens tijdens sequentiële I/O-modus.
begin-adres	lengte	naam	omschrijving
179	2	VRECL	Variabele record-lengte. Standaardwaarde is 128. Instelbaar via argument van OPEN-opdracht.

181	2	PHYREC	Huidig fysiek record-nummer.
183	2	LOGREC	Huidig logisch record-nummer.
185	1	***	Nog niet in gebruik.
186	2	OUTPOS	Alleen voor diskette-bestanden. Uitvoer-positie voor PRINT, INPUT en WRITE.
188	n	FIELD	Gegevensbuffer huidig FIELD. Afmeting bepaald door /s. Bij I/O worden VRECL bytes tussen BUFFER en FIELD uitgewisseld. Gebruik dit adres voor controle van gegevens tijdens random I/O-modus.

Voorbeeld 1:

```
100 X=VARPTR(Y)
```

In variabele X wordt een geheugenadres opgeslagen dat verwijst naar het adres waar variabele Y is opgeslagen.

Voorbeeld 2:

```
10 OPEN "DATA.FIL" AS #1
20 FCBADRES=VARPTR(#1)
30 DATADRES=FCBADRES+188
40 A$ = PEEK(DATADRES)
```

Op regel 20 wordt het startadres van het File Control Block opgeslagen in de variabele FCBADRES.
Op regel 30 wordt het adres van de gegevensbuffer opgeslagen in DATADRES.
Op regel 40 wordt het eerste byte van de gegevensbuffer opgeslagen in A$.

VARPTR$ functie

Functie:
Startadres van een variabele aangeven in de vorm van een teken.

Syntaxis:
VARPTR$(*variabele*)

Opmerkingen:
Het argument *variabele* moet een in het programma gebruikte variabelenaam zijn.

Attentie:
Gebruik VARPTR$ pas om de inhoud van de cel van een matrix op te vragen als u alle andere variabelen al hebt gedeclareerd. De geheugenadressen van matrices veranderen namelijk bij elke toevoeging van nog niet eerder gebruikte variabelen.

VARPTR$ geeft een tekenreeks van 3 bytes:

byte 0	byte 1	byte 2

Byte 0 bevat een van de volgende typen variabelen:
2 geheel getal
3 tekenreeks
4 enkele precisie
8 dubbele precisie

Byte 1 bevat het 8086 adres-format en is het minst-significante byte.
Byte 2 bevat het 8086 adres-format en is het meest-significante byte.

Voorbeeld:

```
100 X=USR(VARPTR$(Y))
```

VIEW opdracht

Functie:
Venster instellen aan de hand van de coördinaten voor de hoeken linksboven en rechtsonder in dat venster.

Syntaxis:
VIEW [[SCREEN] [(*x1,y1*)-(*x2,y2*) [,[*fill*] [,[*kader*]]]]

Opmerkingen:
RUN of VIEW zonder toegevoegde argumenten maken het venster schermvullend.

In argument (*x1,y1*) geeft u de coördinaten op voor de linkerbovenhoek van het venster.
In argument (*x2,y2*) geeft u de coördinaten op voor de rechteronderhoek van het venster.

Met het optionele argument *fill* kunt u het venster laten inkleuren.

Het optionele argument *kader* stelt u in staat een kenmerk op te geven waarmee een kader rond het venster wordt afgebeeld, vooropgesteld dat de afmeting van het gedefinieerde venster daartoe voldoende ruimte overlaat. Als u dit achterwege laat, verschijnt er geen kader.

De x- en y-coördinaten moeten een waarde hebben die bij het gebruikte beeldscherm realiseerbaar is. Via deze coördinaten bepaalt u in welke rechthoek grafische patronen worden afgebeeld. De coördinaten-paren x en y worden zodanig gerangschikt dat de hoogste waarde op de laagste volgt.

Als u het argument [SCREEN] niet gebruikt, wordt de schermpositie voor een af te beelden punt berekend vanuit de cursorpositie. Dit betekent dat de coördinaten (*x1,y1*) worden opgeteld bij de huidige x- en y-coördinaten.

Het aantal x-y paren mag variëren. De enige restrictie is dat x1 niet gelijk mag zijn aan x2 en hetzelfde geldt voor y1 en y2.

Als u wel gebruik maakt van het argument SCREEN, dan wordt bij het berekenen van de schermpositie voor een af te beelden punt alleen uitgegaan van de opgegeven coördinaten x1,y1 en x2,y2. In dat geval kunt u alleen punten laten afbeelden die binnen het venster vallen wat u hebt gedefinieerd.

Als u een venster hebt gedefinieerd, heeft CLS alleen effect binnen dat venster. Wilt u het gehele beeldscherm leegmaken, dan moet u eerst met een nieuwe VIEW-opdracht het venster opheffen en vervolgens CLS gebruiken. CLS stuurt de cursor niet naar de "home"-positie (linksboven). Met CTRL-HOME lukt dat wel.

Voorbeeld 1:

```
VIEW (10,10)-(200,100)
```

Het venster is zodanig gedefinieerd dat de instructie PSET(0,0),3 verwijst naar de schermcoördinaten 10,10 omdat zonder argument SCREEN de coördinaten worden opgeteld.

Voorbeeld 2:

```
VIEW SCREEN (10,10)-(200,100)
```

De VIEW-instructie bevat het argument SCREEN, zodat alleen rekening wordt gehouden met de opgegeven coördinaten. Het door PSET(0,0),3 aangeduide beeldpunt zou niet worden afgebeeld omdat 0,0 buiten het gedefinieerde venster ligt. PSET(10,10),3 ligt binnen het venster.

VIEW PRINT opdracht

Functie:

De afmetingen van een tekstvenster bepalen.

Syntaxis:

VIEW PRINT [*bovenregel* TO *onderregel*]

Opmerkingen:

Als u geen *boven-* en *onderregel* opgeeft, maakt u het gehele beeldscherm tot tekstvenster, d.w.z. de schermregels 1 t/m 24. Het systeem bepaalt automatisch dat regel 25 niet wordt gebruikt.

Instructies als CLS, LOCATE, PRINT en SCREEN hebben na de definitie van een tekstvenster alleen effect binnen dat venster. Hetzelfde geldt voor bepaalde mogelijkheden van de tekst-editor, zoals beeldverschuiving en cursor-verplaatsing.

Lees de beschrijving van VIEW voor meer bijzonderheden.

WAIT opdracht

Functie:

De uitvoering van het programma onderbreken en wachten tot een invoer-poort een bepaalde status heeft.

Syntaxis:

WAIT *poortnummer*, *n*[,*j*]

Opmerkingen:

Voor het *poortnummer* moet een voor het systeem geldig nummer worden gebruikt uit het bereik 0 t/m 65535.

Voor de argumenten *n* en *j* moet een geheel getal worden opgegeven uit het bereik 0-255.

Met behulp van de opdracht WAIT onderbreekt u het programma. Het systeem wacht tot op de opgegeven invoer-poort een bepaald bit-patroon verschijnt.

Op de aan de poort ingelezen gegevens wordt eerst een exclusive OR-bewerking met getal *j* uitgevoerd en vervolgens een AND-bewerking met de waarde *n*. Als dat een uitkomst van 0 oplevert, leest GW-BASIC de invoer-poort opnieuw. Zodra de uitkomst ongelijk is aan 0, hervat het systeem de uitvoering van het BASIC programma met de eerstvolgende opdracht.

WAIT zoekt in byte *n* naar bits die "gezet zijn", d.w.z. "logisch 1" zijn. Als een bit met de waarde 1 wordt gevonden, wordt de uitvoering van het programma hervat. WAIT wacht niet tot een bepaald bit-patroon optreedt, maar zoekt slechts tot één bit met de waarde 1 is gevonden.

U kunt met behulp van de WAIT-opdracht een oneindige lus opstellen, die u kunt afbreken door op CTRL-BREAK te drukken of door een reset op het systeem uit te voeren.

Als u voor argument *j* geen waarde opgeeft, kiest het systeem automatisch de waarde 0.

Voorbeeld:

```
100 WAIT 32,2
```

Het systeem wacht tot op poort 32 de waarde 2 wordt aangeboden.

WHILE...WEND opdracht

Functie:

Herhaald uitvoeren van een reeks instructies, door middel van een programmalus, ook wel "loop" genoemd, zolang een bepaalde voorwaarde waar is.

Syntaxis:

WHILE *expressie*
.
.
. **opdrachten binnen de lus**
.
.
WEND

Opmerkingen:

Zolang de *expressie* waar (true, ongelijk aan 0) is, dan worden de opdrachten binnen de lus uitgevoerd. WEND stuurt het programma terug naar de WHILE-opdracht om na te gaan of nog steeds aan de gestelde voorwaarde wordt voldaan. Als dat zo is, worden de opdrachten opnieuw uitgevoerd.
Zodra niet langer aan de in WHILE gestelde voorwaarde wordt voldaan, gaat de uitvoering van het programma verder met de eerstvolgende opdracht na WEND.

WHILE...WEND-lussen mogen onbeperkt worden "genest", d.w.z. binnen een lus mag een andere lus worden opgenomen. Daarbij geldt dat elke WEND-opdracht refereert aan de laatst uitgevoerde WHILE-opdracht.

Als het programma een WHILE-opdracht uitvoert waarmee geen WEND correspondeert, dan verschijnt de foutmelding "WHILE zonder WEND". Omgekeerd treedt bij uitvoering van een WEND zonder bijbehorende WHILE de fout "WEND zonder WHILE" op.

Voorbeeld:

```
90 'Sorteer matrix A$
100 FLIP=1
110 WHILE FLIP
120 FLIP=0
130 FOR N = 1 TO J-1
130 IF A$(N)>A$(N+1) THEN SWAP A$(N),A$(N+1): FLIP=1
140 NEXT N
150 WEND
```

WIDTH opdracht

Functie:

De regellengte voor beeldscherm en printer instellen.

Syntaxis:

WIDTH *lengte*
WIDTH *bestandsnummer,lengte*
WIDTH "*syselement*",*lengte*

Opmerkingen:

Het argument *lengte* bepaalt het aantal tekens per regel en moet een waarde hebben uit het bereik 0-255.

Het argument *bestandsnummer* verwijst naar het nummer van het geopende bestand.

Argument "*syselement*" bestaat uit een geldige tekenreeks-expressie, die aangeeft op welk onderdeel van het computersysteem de WIDTH-opdracht betrekking heeft. De mogelijkheden zijn: SCRN:, LPT1:, LPT2:, LPT3:, COM1: en COM2:.

Regellengte voor het beeldscherm wijzigen

Met behulp van de twee onderstaande opdrachten stelt u de regellengte op het beeldscherm in. Daarbij zijn alleen de waarden 40 of 80 toegestaan.

```
WIDTH lengte
WIDTH "SCRN:",lengte
```

Zie de beschrijving van SCREEN voor meer bijzonderheden.

Verandering van de scherm-modus beïnvloedt de regellengte alleen als SCREEN 2 wordt gewijzigd in SCREEN 1 of SCREEN 0, of omgekeerd.

Attentie:
Door wijziging van de regellengte wordt het scherm leeggemaakt en verandert de kleur van het kader in zwart.

Regellengte voor de printer wijzigen

De onderstaande WIDTH-opdracht bepaalt een nieuwe regellengte voor de printer. Deze nieuwe lengte wordt opgeslagen, maar heeft niet onmiddellijk effect.

`WIDTH "LPT1:",` *lengte*

Een opdracht als:

`OPEN "LPT1:" FOR OUTPUT AS` *lengte*

herkent de opgeslagen nieuwe waarde en gebruikt die zodra het bestand is geopend:

`WIDTH` *bestandsnummer, lengte*

Als het bestand is geopend voor LPT1:, wordt de regellengte voor de printer direct aangepast aan de nieuwe waarde. Op deze wijze is het mogelijk de regellengte naar eigen wens aan te passen als het bestand is geopend. Deze versie van WIDTH is alleen van belang voor de opgegeven printer. Zodra op een regel het aantal afgedrukte tekens gelijk is aan de opgegeven lengte, sluit GW-BASIC die regel af met een carriage return. Als het aantal af te drukken tekens te groot was voor één regel, worden de resterende tekens op de volgende regel afgedrukt.

De regellengte voor de printer kan worden ingesteld op een waarde uit het bereik 1-255. Als u voor de printer WIDTH 255 opgeeft, activeert u de regelomslag. Dit heeft hetzelfde effect als een onbegrensde regellengte. Zodra u een regellengte opgeeft buiten het geldige bereik, volgt de foutmelding "Onjuiste functieaanroep" en blijft de regellengte ongewijzigd.

Als u WIDTH gebruikt voor de regellengte van communicatie-bestanden, dan wordt bij het bereiken van de regellengte een carriage return ingevoegd. De regellengte heeft geen invloed op de invoer- of uitvoer-buffer.

Voorbeeld:

```
10 WIDTH "LPT1:",75
20 OPEN "LPT1:" FOR OUTPUT AS #1
.
.
.
6000 WIDTH #1,40
```

Regel 10 slaat de nieuwe regellengte op (75 tekens per regel).
Regel 20 opent bestand #1 voor de printer en stelt de regellengte voor PRINT #1-opdrachten in op 75.
Regel 6000 verandert de regellengte voor de printer naar 40 tekens per regel.

WINDOW opdracht

Functie:

Lijnen, en grafische objecten tekenen in een oppervlak dat niet wordt begrensd door de fysieke afmetingen van het scherm.

Syntaxis:

WINDOW [[SCREEN](*x1,y1*)-(*x2,y2*)]

Opmerkingen:

De door u opgegeven coördinaten (*x1,y1*) en (*x2,y2*) worden "wereldcoördinaten" genoemd en zij bestaan uit een willekeurige decimale waarde met enkele precisie. Deze coördinaten bepalen de omvang van een coördinatenruimte die groter mag zijn dan de begrenzing die door de scherm-coördinaten in een VIEW-opdracht werd ingesteld.

WINDOW maakt van uw beeldscherm een rechthoekig venster boven het door de "wereldcoördinaten" bepaalde oppervlak. WINDOW biedt mogelijkheden als zoom en pan. U kunt nu lijnen en grafische objecten tekenen in een ruimte die groter is dan de afmeting van het beeldscherm. GW-BASIC zet de door u opgegeven "wereldcoördinaten" om naar coördinaten die aangepast zijn aan de werkelijke afmeting van het beeldscherm. Daardoor kunnen de in feite te grote tekeningen toch op het scherm worden afgebeeld.

Een WINDOW-opdracht zonder het optionele argument SCREEN inverteert in latere grafische opdrachten de y-coördinaat. De coördinaat (*x1,y1*) verwijst dan naar de hoek linksonder in het scherm, de coördinaat (*x2,y2*) naar de hoek rechtsboven. Op die wijze verkrijgt u een scherm met echte Cartesiaanse coördinaten.

Als u aan een WINDOW-opdracht het argument SCREEN toevoegt, vindt geen inversie plaats. Dan verwijst (*x1,y1*) naar linksboven en (*x2,y2*) naar rechtsonder.

Een WINDOW-opdracht rangschikt de coördinaten zodanig dat de laagste waarden voorafgaan aan de hoogste. Twee voorbeelden:

```
WINDOW (50,50)-(10,10)
```

wordt gewijzigd in:

```
WINDOW (10,10)-(50,50)
```

```
WINDOW (-2,2)-(2,-2)
```

wordt gewijzigd in:

```
WINDOW (-2,-2)-(2,2)
```

Voor de coördinaten zijn alle waarden toegestaan, met dien verstande dat x1 en x2 nooit aan elkaar gelijk mogen zijn en hetzelfde geldt voor y1 en y2.

De opdracht WINDOW zonder nadere toevoegingen heft het effect van eerdere WINDOW-opdrachten op.

Voorbeeld 1:
Stel dat u de volgende instructie intikt:

```
NEW
SCREEN 2
```

Dan worden de volgende standaardcoördinaten geactiveerd:

```
0,0320,0639,0

\/y neemt toe
320,100

0,199320,100639,199
```

Voorbeeld 2:
U gebruikt de instructie

```
WINDOW (-1,-1)-(1,1)
```

Dan activeert u de Cartesiaanse coördinaten die beschreven worden door:

```
-1,10,11,1
/\y neemt toe

0,0

\/y neemt af
-1,10,11,1
```

Voorbeeld 3:
Door de instructie

```
WINDOW SCREEN (-1,-1)-(1,1)
```

activeert u de niet-geïnverteerde coördinaat die wordt gedefinieerd door:

```
-1,-10,-11,-1
/y neemt af

0,0

\/y neemt toe
-1,10,11,1
```

De instructies RUN, SCREEN en WINDOW zonder nadere argumenten schakelen het effect van een met WINDOW gedefinieerd stelsel "wereldcoördinaten" uit. Vanaf dat moment geldt weer de fysieke begrenzing van het beeldscherm.

WRITE opdracht

Functie:
Gegevens op het scherm afbeelden.

Syntaxis:
WRITE [*expressielijst*]

Opmerkingen:

Als u de optionele *expressielijst* achterwege laat, verschijnt een lege regel. Voegt u wel expressies toe, dan wordt de inhoud of waarde daarvan op het scherm weergegeven. U kunt zowel tekenreeksen als numerieke expressies opgeven. De expressies of variabelenamen moeten van elkaar worden gescheiden door komma's of puntkomma's.

De op het scherm afgebeelde gegevens worden van elkaar gescheiden door komma's en letterlijke tekst staat tussen aanhalingstekens ("). Na het laatste gegeven gaat de cursor naar de eerstvolgende schermregel.

Numerieke waarden worden door WRITE met dezelfde layout afgebeeld als door een PRINT-opdracht.

Voorbeeld:

```
10 A=80:B=90:C$="Dit is het laatste gegeven"
20 WRITE A,B,C$
RUN
80,90,"Dit is het laatste gegeven"
Ok
```

WRITE# opdracht

Functie:

Gegevens opslaan in een sequentieel bestand.

Syntaxis:

WRITE #*bestandsnummer*, *expressielijst*

Opmerkingen:

Het *bestandsnummer* is het nummer dat werd toegekend toen het bestand werd geopend.

In de *expressielijst* staan een of meer tekenreeksen en/of numerieke expressies, van elkaar gescheiden door komma's.

WRITE# is niet hetzelfde als PRINT#. WRITE# plaatst tussen de afzonderlijke gegevens een komma en plaatst tekst-tekenreeksen tussen aanhalingstekens, zodat u zelf in de expressielijst geen scheidingstekens hoeft op te geven. Bovendien plaatst WRITE# voor een positief getal geen lege tekenpositie. Na het laatste gegeven voegt WRITE# automatisch een carriage return/line feed toe.

Voorbeeld:

Stel dat u ergens in het programma hebt aangegeven: A$="camera" en ook:

```
B$="93604-1".
```

Dan zou de opdracht

```
WRITE#1,A$,B$
```

op de diskette in het bestand met bestandsnummer 1 het volgende opslaan:

```
"camera", "93604-1"
```

Een later volgende INPUT$ opdracht, bijvoorbeeld:

```
INPUT#1,A$,B$
```

slaat dan "camera" in A$ op en "93604-1" in B$.

APPENDIX A
FOUTCODES EN FOUTMELDINGEN

1 NEXT zonder FOR

Bij een NEXT-opdracht ontbreekt de FOR-opdracht. Controleer of de variabele van de FOR-opdracht overeenkomt met de variabele van de NEXT-opdracht.

2 Syntaxisfout

Een regel bevat een onjuiste combinatie van tekens (zoals een haakje te veel of te weinig, spelfouten in commando's of opdrachten, onjuiste interpunctie enz.). Als gevolg van deze fout geeft GW-BASIC in de edit-modus de regel weer waarin de fout staat.

3 RETURN zonder GOSUB

Het programma bevat een RETURN-opdracht waaraan geen GOSUB is voorafgegaan.

4 Geen DATA meer

Een READ-opdracht wordt uitgevoerd terwijl het programma geen DATA-opdrachten meer bevat die nog ingelezen kunnen worden.

5 Onjuiste functieaanroep

Een parameter met een te hoge waarde wordt overgedragen aan een wiskundige functie of een tekenreeks-functie. Deze foutmelding verschijnt ook bij:

- een negatieve of extreem grote matrix-dimensie
- een negatief argument of een 0 in een LOG-functie
- een negatief getal bij de functie SQR
- een negatieve mantisse met een macht die bestaat uit een niet-geheel getal
- het oproepen van een USR-functie waarvoor het startadres niet is opgegeven
- een onjuist argument voor MID$, LEFT$, RIGHT$, INP, OUT, WAIT, PEEK, POKE, TAB, SPC, STRING$, SPACE$, INSTR, of ON...GOTO.

6 Te groot getal

Het resultaat van een berekening is te groot voor GW-BASIC. Bij "underflow", een getal dat te klein is, wordt de uitkomst 0 en wordt de berekening zonder verdere foutmeldingen uitgevoerd.

7 Geen geheugenruimte meer

Het programma is te lang, heeft te veel FOR...NEXT-lussen, GOSUB-opdrachten, of te complexe expressies of variabelen. Gebruik CLEAR om meer stack- of geheugenruimte vrij te maken.

8 Ongedefinieerd regelnummer

GOTO, GOSUB, IF...THEN...ELSE of DELETE verwijzen naar een niet bestaand regelnummer.

9 Te hoog indexgetal

Er wordt verwezen naar een matrix-cel met een nummer dat groter is dan voor die dimensie is opgegeven of met een onjuist aantal dimensies.

10 Dubbele definitie

Voor dezelfde matrix worden twee DIM-opdrachten opgegeven of er is een DIM-opdracht opgegeven voor een matrix met de standaarddimensie 10.

11 Deling door nul

In een uitdrukking komt een deling door nul voor, of het resultaat van een bewerking is nul en daarop vindt een machtsverheffing plaats met een negatieve exponent. Het grootst mogelijke getal, voorzien van het teken van het deeltal (in het geval van deling door nul) of het grootst mogelijke positieve getal (in het geval van machtsverheffing) wordt als uitkomst aangenomen en het programma wordt vervolgd.

12 Ongeldig in directe modus

In de commando-modus is een opdracht gebruikt die tijdens die modus niet is toegestaan.

13 Onjuist type

Aan een tekenreeks-variabele wordt een numerieke waarde toegekend of omgekeerd. Bij een functie waar een numeriek argument wordt verwacht, is een tekenreeks ingetikt, of omgekeerd.

14 Geen ruimte meer voor karakterreeksen

Het aantal tekenreeks-variabelen is te groot voor de geheugenruimte van GW-BASIC. GW-BASIC reserveert dynamisch geheugenruimte tot het gehele geheugen vol is.

15 Karakterreeks te lang

De maximale tekenreeks-lengte is 255 tekens.

16 Karakterreeks te complex

Een tekenreeks-expressie is te lang of te complex. Deel de expressie op in kleinere expressies.

17 Kan programma niet vervolgen

Er wordt geprobeerd een programma verder uit te voeren dat:

- onderbroken is na een fout
- gewijzigd is tijdens een onderbreking
- niet bestaat.

18 Ongedefinieerde gebruikersfunctie

Er wordt een USR-routine opgeroepen die nog niet met DEF is gedefinieerd.

19 Geen RESUME

Een routine voor foutafhandeling bevat geen RESUME-opdracht.

20 RESUME zonder fout

Er is een RESUME-opdracht uitgevoerd zonder dat vooraf een fout is opgetreden.

21 Niet afdrukbare fout

Voor de betreffende fout bestaat geen foutmelding. Dit hangt meestal samen met een fout waarvoor geen foutcode is gedefinieerd.

22 Ontbrekende operand

Expressie bevat operator, maar geen getal of variabelenaam.

23 Regelbuffer vol

De ingevoerde regel is te lang.

24 Apparaat niet beschikbaar

GW-BASIC heeft binnen een vooraf bepaalde tijd geen informatie ontvangen van een randapparaat.

25 Onjuist apparaat

Duidt op een hardware-fout in de printer of het interface.

26 FOR zonder NEXT

Bij een FOR-opdracht ontbreekt een NEXT.

27 Geen papier

De printer bevat geen papier meer, of er is een fout in de printer opgetreden.

28 Niet afdrukbare fout

Voor de betreffende fout bestaat geen foutmelding. Dit hangt meestal samen met een fout waarvoor geen foutcode is gedefinieerd.

29 WHILE zonder WEND

Bij een WHILE-opdracht ontbreekt WEND.

30 WEND zonder WHILE

Bij een WEND-opdracht ontbreekt WHILE.

31-49 Niet afdrukbare fout

Voor de betreffende fout bestaat geen foutmelding. Dit hangt meestal samen met een fout waarvoor geen foutcode is gedefinieerd.

50 Field-lengte te groot

De FIELD-opdracht vereist meer bytes dan bij de gedefinieerde record-lengte van het onbepaald bestand mogelijk is.

51 Interne fout

Er is een interne fout opgetreden in GW-BASIC. Leg het probleem voor aan uw dealer.

52 Onjuist bestandsnummer

Een opdracht of commando verwijst naar een bestandsnummer dat niet geopend is of naar een bestandsnummer met een waarde die groter is dan bij het initialiseren is opgegeven.

53 Bestand niet gevonden

LOAD, KILL, NAME, of OPEN verwijst naar een bestand dat niet aanwezig is op de huidige diskette.

54 Onjuiste bestandsnummer

PUT, GET, of LOF zijn gebruikt voor een sequentieel bestand, LOAD werd gebruikt voor een onbepaald bestand of in een OPEN-opdracht werd een andere modus dan I, O, R of A opgegeven.

55 Bestand reeds open

Voor een reeds geopend bestand wordt een OPEN- of KILL-opdracht gegeven.

56 Niet afdrukbare fout

Voor de betreffende fout bestaat geen foutmelding. Dit hangt meestal samen met een fout waarvoor geen foutcode is gedefinieerd.

57 Invoer/Uitvoer fout

Duidt meestal op een fout tijdens I/O-operaties met een station, maar kan ook betrekking hebben op andere invoer- en uitvoeronderdelen. Deze fout kan niet via het besturingssysteem worden hersteld.

58 Bestand bestaat reeds

De bestandsnaam die in een NAME-opdracht is opgegeven, is al voor een ander bestand gebruikt.

59-60 Niet afdrukbare fout

Voor de betreffende fout bestaat geen foutmelding. Dit hangt meestal samen met een fout waarvoor geen foutcode is gedefinieerd.

61 Schijf is vol

Alle opslagruimte op de schijf is gebruikt.

62 Bestandseinde gepasseerd

Een INPUT-opdracht is uitgevoerd nadat alle gegevens uit het bestand al zijn ingelezen of wordt gebruikt bij een leeg bestand. U kunt deze fout voorkomen door gebruik te maken van de EOF-functie.

63 Onjuist bloknummer

Het record-nummer dat bij PUT of GET wordt gebruikt is hoger dan het toegestane maximum (16.777.215), of is gelijk aan nul.

64 Ongeldige bestandsnaam

Bij LOAD, SAVE, KILL of OPEN wordt een onjuiste bestandsnaam gebruikt (bijvoorbeeld een te lange bestandsnaam).

65 Niet afdrukbare fout

Voor de betreffende fout bestaat geen foutmelding. Dit hangt meestal samen met een fout waarvoor geen foutcode is gedefinieerd.

66 Directe opdracht gevonden

Bij het laden van een bestand in ASCII-formaat is een opdracht aangetroffen die alleen in de commando-modus is toegestaan en daarom wordt het laden afgebroken.

67 Te veel bestanden

Er is geprobeerd een nieuw bestand aan te maken (met SAVE of OPEN) met onjuiste bestandsspecificaties of terwijl alle directories vol zijn.

68 Apparaat niet beschikbaar

Er is een bestand geopend voor een niet bestaand apparaat. Dit houdt in dat het gewenste apparaat niet bestaat, zoals LPT2: of LPT3:, of dat het apparaat door de gebruiker ontoegankelijk werd gemaakt. Dit probleem treedt op bij het uitvoeren van een OPEN "COM1:"-opdracht, terwijl de gebruiker eerder op de commandoregel de optie /c: heeft opgegeven, waardoor geen communicatie via de RS-232-poorten mogelijk is.

69 Communicatiebuffer vol

Deze melding verschijnt bij het opgeven van een opdracht voor invoer via de communicatiepoort terwijl de wachtrij voor invoer al vol is. Gebruik in dit geval ON ERROR GOTO om de opdracht opnieuw uit te voeren. Direct daaropvolgende invoeropdrachten kunnen het probleem oplossen, tenzij tekens sneller worden ingelezen dan zij door het programma kunnen worden verwerkt. In dit geval kunt u het probleem op verschillende manieren oplossen:

- maak de COM ontvangstbuffer groter met behulp van de optie /c:
- geef opdracht voor een handshake-protocol met de host/satellite (bijvoorbeeld XON/XOFF in het TTY-voorbeeld) om het invoeren van gegevens regelmatig te onderbreken
- gebruik een lagere transmissiesnelheid (baud rate) voor het ontvangen en verzenden van gegevens.

70 Schijf heeft schrijfbeveiliging

Dit is een van de drie fouten die door de disk-controller worden gedetecteerd. Bovenstaande melding verschijnt in de volgende gevallen:

- bij het schrijven op een tegen overschrijven beschermde diskette
- bij het aanroepen van een bestand dat al in gebruik is door een ander programma
- als bij een UNLOCK-opdracht een andere reeks records is opgegeven dan bij de corresponderende LOCK-opdracht.

71 Schijf niet gereed

Het station is niet vergrendeld of het station bevat geen diskette. Gebruik ON ERROR GOTO om deze fout te herstellen.

72 Schijffout

De diskette-controller heeft een fout ontdekt in de hardware of het geheugenmedium zelf. Meestal duidt dit op een beschadiging in het magnetisch materiaal. Kopieer de bestanden naar een nieuwe diskette en formatteer de beschadigde diskette opnieuw. Door het formatteren worden de slechte sporen opgespoord en in een speciale tabel opgeslagen. De rest van de diskette is nu weer gewoon bruikbaar.

73 Functie in uitgebreide BASIC

Er werd gebruik gemaakt van een gereserveerd woord dat in deze versie van GW-BASIC niet beschikbaar is.

74 Hernoemd naar andere schijf

Bij het herbenoemen van een bestand is een ander station opgegeven dan dat waarin de diskette zich bevond waarop het bestand onder de oude naam was opgeslagen. De nieuwe naam wordt niet toegekend.

75 Pad/bestand niet toegankelijk

Bij het uitvoeren van OPEN, MKDIR, CHDIR of RMDIR, kon MS-DOS de juiste koppeling tussen opgegeven pad en bestand niet leggen. De bewerking wordt niet verder uitgevoerd.

76 Pad niet gevonden

Bij het uitvoeren van OPEN, MKDIR, CHDIR of RMDIR, kon MS-DOS het opgegeven pad niet vinden. De bewerking wordt niet verder uitgevoerd.

APPENDIX B
WISKUNDIGE FUNCTIES

Wiskundige functies die niet in GW-BASIC voorkomen, kunnen als volgt worden berekend:

Secans:
```
SEC(X)=1/COS(X)
```
Cosecans:
```
CSC(X)=1/SIN(X)
```
Cotangens:
```
COT(X)=1/TAN(X)
```
Inverse sinus:
```
ARCSIN(X)=ATN(X/SQR(-X*X+1))
```
Inverse cosinus:
```
ARCCOS(X)=-ATN(X/SQR(-X*X+1))+1.5708
```
Inverse secans:
```
ARCSEC(X)=ATN(X/SQR(X*X-1))+SGN(SGN(X)-1)*1.5708
```
Inverse cosecans:
```
ARCCSC(X)=ATN(X/SQR(X*X-1))+(SGN(X)-1)*1.5708
```
Inverse cotangens:
```
ARCCOT(X)=ATN(X)+1.5708
```
Hyperbolische sinus:
```
SINH(X)=(EXP(X)-EXP(-X))/2
```
Hyperbolische cosinus:
```
COSH(X)=(EXP(X)+EXP(-X))/2
```
Hyperbolische tangens:
```
TANH(X)=(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
```
Hyperbolische secans:
```
SECH(X)=2/(EXP(X)+EXP(-X))
```
Hyperbolische cosecans:
```
CSCH(X)=2/(EXP(X)-EXP(-X))
```
Hyperbolische cotangens:
```
COTH(X)=(EXP(X)+EXP(-X))/(EXP(X)-EXP(-X))
```
Inverse hyperbolische sinus:
```
ARCSINH(X)=LOG(X/SQR(X*X+1))
```
Inverse hyperbolische cosinus:
```
ARCCOSH(X)=LOG(X+SQR(X*X-1))
```
Inverse hyperbolische tangens:
```
ARCTANH(X)=LOG((1+X)/(1-X))/2
```
Inverse hyperbolische secans:
```
ARCSECH(X)=LOG((SQR(-X*X+1)+1)/X)
```
Inverse hyperbolische cosecans:
```
ARCCSCH(X)=LOG((SGN(X)*SQR(X*X+1)+1)/X)
```
Inverse hyperbolische cotangens:
```
ARCCOTH(X)=LOG((X+1)/(X-1))/2
```

APPENDIX C
ASCII-CODES

Decimaal	Octaal	Hexadecimaal	Teken
000	000	00H	NUL
001	001	01H	SOH
002	002	02H	STX
003	003	03H	ETX
004	004	04H	EOT
005	005	05H	ENQ
006	006	06H	ACK
007	007	07H	BEL
008	010	08H	BS (Backspace)
009	011	09H	HT
010	012	0AH	LF (Linefeed)
011	013	0BH	VT
012	014	0CH	FF (Formfeed)
013	015	0DH	CR (Carriage return)
014	016	0EH	SO
015	017	0FH	SI
016	020	10H	DLE
017	021	11H	DC1
018	022	12H	DC2
019	023	13H	DC3
020	024	14H	DC4
021	025	15H	NAK
022	026	16H	SYN
023	027	17H	ETB
024	030	18H	CAN
025	031	19H	EM
026	032	1AH	SUB
027	033	1BH	ESC (Escape)
028	034	1CH	FS
029	035	1DH	GS
030	036	1EH	RS
031	037	1FH	US
032	040	20H	SPACE
033	041	21H	!
034	042	22H	"
035	043	23H	#
036	044	24H	$
037	045	25H	%
038	046	26H	&
039	047	27H	'
040	050	28H	(
041	051	29H	)
042	052	2AH	*
043	053	2BH	+
044	054	2CH	,

Decimaal	Octaal	Hexadecimaal	Teken
045	055	2DH	-
046	056	2EH	.
047	057	2FH	/
048	060	30H	0
049	061	31H	1
050	062	32H	2
051	063	33H	3
052	064	34H	4
053	065	35H	5
054	066	36H	6
055	067	37H	7
056	070	38H	8
057	071	39H	9
058	072	3AH	:
059	073	3BH	;
060	074	3CH	<
061	075	3DH	=
062	076	3EH	>
063	077	3FH	?
064	100	40H	@
065	101	41H	A
066	102	42H	B
067	103	43H	C
068	104	44H	D
069	105	45H	E
070	106	46H	F
071	107	47H	G
072	110	48H	H
073	111	49H	I
074	112	4AH	J
075	113	4BH	K
076	114	4CH	L
077	115	4DH	M
078	116	4EH	N
079	117	4FH	O
080	120	50H	P
081	121	51H	Q
082	122	52H	R
083	123	53H	S
084	124	54H	T
085	125	55H	U
086	126	56H	V
087	127	57H	W
088	130	58H	X
089	131	59H	Y
090	132	5AH	Z
091	133	5BH	[
092	134	5CH	\
093	135	5DH	]
094	136	5EH	^
095	137	5FH	_
096	140	60H	`
097	141	61H	a
098	142	62H	b
099	143	63H	c
100	144	64H	d
101	145	65H	e
102	146	66H	f
103	147	67H	g
104	150	68H	h
105	151	69H	i

Decimaal	Octaal	Hexadecimaal	Teken
106	152	6AH	j
107	153	6BH	k
108	154	6CH	l
109	155	6DH	m
110	156	6EH	n
111	157	6FH	o
112	160	70H	p
113	161	71H	q
114	162	72H	r
115	163	73H	s
116	164	74H	t
117	165	75H	u
118	166	76H	v
119	167	77H	w
120	170	78H	x
121	171	79H	y
122	172	7AH	z
123	173	7BH	{
124	174	7CH	|
125	175	7DH	}
126	176	7EH	~
127	177	7FH	DEL (Rubout)

APPENDIX D
SUBROUTINES IN ASSEMBLEERTAAL

Deze appendix is in de eerste plaats geschreven voor gebruikers die al enige ervaring hebben in het programmeren in assembleertaal.

Met behulp van de USR-functie en de CALL-opdracht kunt u gebruik maken van subroutines geschreven in assembleertaal.

Met behulp van de USR-functie kunt u assembleerroutines op dezelfde wijze oproepen als functies van GW-BASIC. Het is echter aan te bevelen de CALL-opdracht te gebruiken voor het laden van machinetaal-programma's vanuit GW-BASIC. Het voordeel van CALL ten opzichte van de USR-functie is dat CALL compatibel is met meer talen, beter leesbare codes produceert en gebruik kan maken van meerdere argumenten.

RESERVEREN VAN GEHEUGENRUIMTE

Voordat u een assembleer- of machinetaalroutine kunt laden, moet u geheugenruimte vrijmaken. Er zijn drie manieren om geheugenruimte te reserveren voor assembleerroutines:

- Geef een matrix op en gebruik VARPTR voor het opsporen van het startadres van de matrix voordat u routines aanroept.
- Gebruik de optie /m op de commandoregel. Roep het Data Segment (DS) van GW-BASIC op en wijs de grootte van DS toe aan de gereserveerde ruimte boven het data-segment.
- Voer een .COM-bestand uit dat blijft opgeslagen en sla hierin een pointer op in een niet-gebruikte interrupt-vector locatie.

Er zijn drie manieren om assembleerroutines te laden:

- Laad het bestand via een BLOAD-opdracht in het geheugen. Gebruik vervolgens DEBUG om het bestand in een .EXE-bestand in een hoog geheugenadres te laden. Roep GW-BASIC op en sla het .EXE-bestand op met behulp van BSAVE.
- Roep een .COM-bestand op dat de routines bevat. Sla de pointer voor deze routines op in niet-gebruikte interrupt-vector locaties zodat uw GW-BASIC programma de pointer kan vinden en op die wijze ook de routines kan gebruiken.
- Sla de routine op in het eerder gespecificeerde gebied.

Als voor het oproepen van een assembleerroutine meer stack-ruimte nodig is, kunt u stack-ruimte voor GW-BASIC verkleinen en een nieuwe stack opstellen voor de routines. Alvorens u echter vanuit de subroutine terugkeert, moet de stack-ruimte van GW-BASIC weer worden ingesteld op de oude omvang.

CALL-OPDRACHT

Syntaxis:
CALL *variabelenaam[argumenten]*

Variabelenaam bevat het startadres van de opgeroepen subroutine, in het huidige segment.

Argumenten zijn de variabelen of constanten, gescheiden door komma's, die aan de subroutine worden doorgegeven.

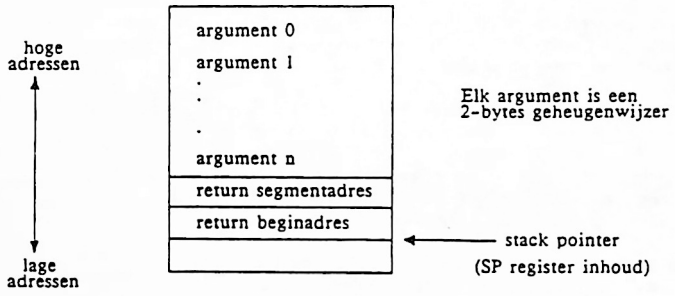
Voor elke argument wordt het uit 2 bytes bestaande startadres van de locatie binnen het data-segment (DS) in de stack geplaatst.

Het code-segment (CS) en het startadres (IP) voor de terugkeer naar GW-BASIC worden in de stack geplaatst.

De programmauitvoering wordt overgedragen aan de subroutine. Deze subroutine bevindt zich in het segment dat in de laatste DEF SEG-opdracht is aangeduid, beginnend op het startadres dat in de variabelenaam is opgegeven.

Het stack-segment (SS), data-segment (DS), extra segment (ES) en de stack pointer (SP) moeten ongewijzigd blijven.

Onderstaand overzicht geeft de status aan van de stack tijdens de CALL-opdracht:

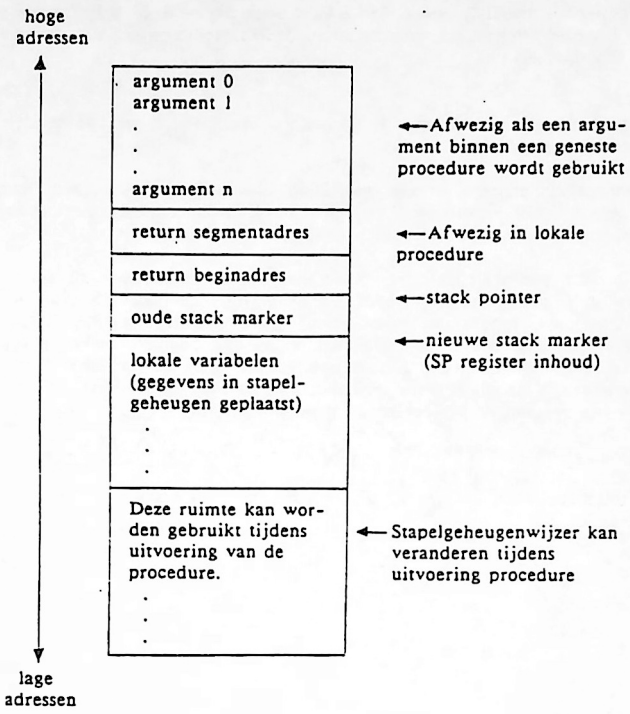


De routine bestuurt nu de programma-uitvoering. U kunt verwijzen naar parameters door de stack pointer (SP) in te stellen op de base pointer (BP) en aan BP een positieve drempelwaarde toe te voegen.

Bij het overgaan naar de routine verwijzen de segment-registers DS, ES en SS naar het segmentadres waar de GW-BASIC interpreter-code is opgeslagen. Het register voor het code-segment (CS) bevat de waarde die in de laatst uitgevoerde DEF SEG-opdracht is opgegeven. Als geen DEF SEG werd uitgevoerd, dan verwijst CS naar de zelfde adressen als DS, ES en SS (standaardwaarde van DEF SEG).

Het onderstaande schema toont de status van de stack tijdens de uitvoering van de opgeroepen routine:

Opzet van de stack tijdens het uitvoeren van een CALL-opdracht:



Bij het coderen van een subroutine moet u rekening houden met de volgende zeven voorschriften:

1 De opgeroepen routine kan de inhoud van de registers AX, BX, CX, DX, SI, DI en BP vernietigen. U hoeft deze registers niet te herstellen voordat u terugkeert naar GW-BASIC. Alle segmentregisters en de stack pointer moeten echter wel worden hersteld. Het is bovendien aan te bevelen de in- of uitgeschakelde interrupts te herstellen in de oorspronkelijke staat.

2 Het aangeroepen programma moet het aantal en de lengte van de opgegeven variabelen kennen. Verwijzingen naar parameters zijn positieve waarden, toegevoegd aan BP, waarbij wordt aangenomen dat de aangeroepen routine de huidige stack pointer met behulp van MOV BP,SP naar BP heeft verplaatst. Als drie variabelen zijn doorgegeven, dan worden de adressen van P0, P1 en P2 respectievelijk BP+10, BP+8 en BP+6.

3 De aangeroepen routine moet een RETURN n (n is twee maal het aantal parameters in de argumentenlijst) uitvoeren om de stack aan te passen aan het begin van de aanroep-bewerking. Bovendien moeten programma's worden gedefinieerd met een PROC FAR-opdracht.

4 Als u in de argumentenlijst opgeeft in welke variabelenaam de resultaten moeten worden opgeslagen, worden deze aan GW-BASIC doorgegeven.

5 Als het argument een tekenreeks is, verwijst de startwaarde voor parameters naar drie bytes die samen de "string descriptor" worden genoemd. Byte 0 van de string descriptor bevat de lengte van de tekenreeks (0 tot 255). Bytes 1 en 2 zijn respectievelijk de acht laagste en acht hoogste bits van het startadres van de tekenreeks in het data-segment van GW-BASIC.

Opmerking:
De aangeroepen routine mag de inhoud van deze drie bytes van de string descriptor niet wijzigen.

6 Tekenreeksen kunnen worden gewijzigd door gebruikers-routines, maar de lengte moet ongewijzigd blijven. GW-BASIC kan niet werken met tekenreeksen waarvan de lengte is gewijzigd door externe routines.

7 Als u voor een argument een tekenreeks tussen aanhalingstekens opgeeft, verwijst de string descriptor naar een programmatekst. Zorg ervoor dat u hierdoor uw programma niet wordt gewijzigd of gewist. U kunt onvoorspelbare resultaten vermijden door +"" toe te voegen aan de tekenreeks. Bijvoorbeeld: de onderstaande regel geeft aan dat de tekenreeks moet worden gekopieerd naar de tekenreeks-ruimte buiten de voor het programma gereserveerde geheugenruimte:

```
20 A$="BASIC"+""
```

De tekenreeks kan nu worden gewijzigd zonder dat het programma wordt verminkt.

Voorbeeld:

```
100 DEF SEG=&H2000
110 ACC=&H7FA
120 CALL ACC(A,B$,C)
.
.
.
```

Regel 100 stelt de waarde voor segment in op het hexadecimale adres 2000. De waarde van variabele ACC wordt toegevoegd aan het adres. Daarvoor wordt op het laagste woord achter de DEF SEG-waarde een shift-bewerking uitgevoerd, waarbij de bits vier posities naar links worden verschoven. (Dit wordt door de processor uitgevoerd; het is geen GW-BASIC instructie.) Hier wordt ACC op &H7FA ingesteld, zodat bij het aanroepen van ACC de routine wordt uitgevoerd die begint op het hexadecimale adres 2000:7FA.

Onmiddellijk na het aanroepen van de routine blijven slechts 16 bytes (acht woorden) beschikbaar binnen de gereserveerde stack-ruimte. Als het opgeroepen programma meer stack-ruimte nodig heeft, moet het gebruikersprogramma extra ruimte toekennen aan de stack pointer. Vergeet niet de stack pointer weer op de oorspronkelijke waarde in te stellen voor u terugkeert naar GW-BASIC.

Het onderstaande assembleerprogramma demonstreert het gebruik van de doorgegeven parameters en de opslag van het resultaat in variabele C.

N.B. De routine moet het variabeletype van de doorgegeven parameters kennen. In onderstaand voorbeeld kopieert MOVSW slechts twee bytes. Dit is voldoende als de variabelen A en C gehele getallen zijn. Voor variabelen van enkele of dubbele precisie zouden respectievelijk vier of acht bytes moeten worden gekopieerd.

```
MOV BP,SP          ;slaat waarde van SP op in BP
MOV BX,8[BP]       ;slaat adres van B$ op in CL
MOV CL,[BX]        ;slaat lengte van B$ op in CL
MOV DX,1[BX]       ;slaat adres van tekst van B$ op in DX
MOV SI,10 [BP]     ;slaat adres van A op in SI
MOV DI,6[BP]       ;slaat naar C verwijzende pointer op in DI
MOVSW              ;slaat variabele A op in C
RET 6              ;slaat stack op en keert terug naar BASIC
```

USR-FUNCTIE

Hoewel de CALL-opdracht optimaal geschikt is voor het aanroepen van assembleerroutines, kunt u daarvoor ook de USR-functie gebruiken als de compatibiliteit met oudere programma's dat vereist.

Syntaxis:

USR[*n*](*argument*)

Het optionele argument [*n*] is een getal tussen 0 en 9 waarmee de opgeroepen USR-routine wordt opgegeven (zie DEF USR-opdracht). Als u geen waarde voor [*n*] opgeeft, kiest het systeem automatisch USR0.

Voor (*argument*) kunt u een willekeurige numerieke expressie of een tekenreeks opgeven.

Om te zorgen dat het code-segment verwijst naar de aangeroepen routine moet in GW-BASIC voor elke USR-functie een DEF SEG-opdracht worden uitgevoerd. Het in de DEF SEG-opdracht opgegeven segment-adres bepaalt het startadres van de routine.

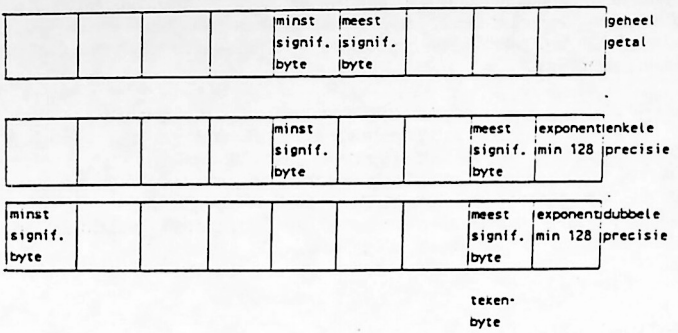
Elke USR-functie moet corresponderen met een DEF USR-opdracht, waarmee eerst de startwaarde voor het aanroepen is opgegeven. Die startwaarde en het op dat moment geactiveerde segment-adres uit de DEF SEG-opdracht bepalen het startadres van de aangeroepen routine.

Nadat een USR-functie is opgegeven, bevat register AL de waarde van de Number Type Flag (NTF) waardoor het type van het opgegeven argument wordt bepaald. NTF kan een van de volgende waarden hebben:

NTF-waarde	duidt op
2	een geheel getal van twee bytes (two's complement format)
3	een tekenreeks
4	een decimaal getal met enkele precisie
8	een decimaal getal met dubbele precisie

Als het argument van een USR-functie een nummer is (AL<>73) wordt de waarde van dit argument geplaatst in de Floating Point Accumulator (FAC). De FAC heeft een lengte van acht bytes en is opgeslagen in het data-segment van GW-BASIC. Register BX verwijst naar het vijfde byte van de FAC. Het volgende schema toont aan hoe de verschillende typen getallen door de Floating Point Accumulator worden verwerkt:

Getaltypen in de "Floating Point Accumulator":



Bij een decimaal getal met enkele precisie:

- bevat BX+3 de exponent min 128. De decimale punt staat links van het meest significante bit van de mantisse.
- bevat BX+2 de hoogste zeven bits van de mantisse, waarbij een 1 aan het begin wordt genegeerd (geïmpliceerd). Bit 7 is het teken van het getal (0=positief, 1=negatief).
- bevat BX+1 de middelste acht bits van de mantisse.
- bevat BX+0 de laagste acht bits van de mantisse.

Bij een geheel getal:

- bevat BX+1 de hoogste acht bits.
- bevat BX+0 de laagste acht bits.

Bij een decimaal getal met dubbele precisie:

- geldt voor BX tot en met BX+3 hetzelfde als voor een decimaal getal met enkele precisie.
- bevatten BX-1 tot BX-4 vier extra bytes van de mantisse. BX-4 bevat de laagste acht bits van de mantisse.

Bij een tekenreeks (aangegeven door de waarde 3 in het AL-register) verwijst het registerpaar DX naar drie bytes (string descriptor). Byte 0 van de string descriptor bevat de tekenreeks-lengte (0 tot 255). Bytes 1 en 2 zijn respectievelijk de acht laagste en de acht hoogste bits van het startadres van de tekenreeks in het data-segment van GW-BASIC.

Als het argument bestaat uit een tekenreeks tussen aanhalingstekens, verwijst de string descriptor naar de programmatekst. Zorg dat u hierdoor het programma niet wijzigt of vernietigt (zie de CALL-opdracht).

Gewoonlijk komt het type (geheel getal, tekenreeks, enkele precisie of dubbele precisie) van de uitkomst van een USR-routine overeen met het variabeletype dat aan die routine werd doorgegeven. In de USR-routine moeten dezelfde registers bewaard blijven als in de CALL-opdracht.

De USR-routine wordt verlaten door een "far return" en de uitkomst moet worden opgeslagen in de FAC.

PROGRAMMA'S VOOR HET OPROEPEN VAN ASSEMBLEERROUTINES

Deze paragraaf bevat twee voorbeeldprogramma's:

- die een assembleerroutine laden voor het optellen van twee getallen
- die de som in het geheugen opslaan en
- die ook zelf in het geheugen blijven opgeslagen.

Het code-segment en het startadres van de eerste routine worden opgeslagen in de interrupt vector op 0:100H.

Het programma in voorbeeld 1 roept een assembleerroutine op.

Voorbeeld 1:

```
10 DEF SEG=0
100 CS=PEEK(&H102)+PEEK(&H103)*256
200 START=PEEK(&H100)+PEEK(&H101)*256
250 DEF SEG
300 C1%=2:C2%=3:C3%=0
400 SOM=START
500 DEF SEG=CS
600 CALL SOM(C1%,C2%,C3%)
700 PRINT C3%
800 END
```

De assembleerroutine die in het bovenstaand programma wordt aangeroepen, moet worden geassembleerd, ge-linked en geconverteerd naar een .COM-bestand. Als die routine wordt uitgevoerd vóór het GW-BASIC programma, dan blijft de routine in het geheugen opgeslagen tot het systeem wordt uitgeschakeld of opnieuw wordt opgestart.

```
0100                    org 100H
0100                    double  segment
                        assume  cs:double
0100    EB 17 90        start:      jmp             start1
0103                    usrprg  proc  far
0103    55              push bp
0104    8B EC           mov bp,sp
0106    8B 76 08        mov si,[bp]+8               ;haal adres van b op
0109    8B 04           mov ax,[si]                 ;haal waarde van b op
010B    8B 76 0A        move si,[bp]+10             ;haal adres van a op
010E    03 04           add ax,[si]                 ;tel a en b op
0110    8B 7E 06        mov di,[bp]+6               ;haal adres van c op
0113    89 05           mov di,ax                   ;sla som op in c
0115    5D              pop bp
0116    CA 00 06        ret 6
0119                    usrprg   endp
```

```
                                                ;
                                                ;Programma voor opslaan
                                                ;van procedure.
                                                ;Startwaarde en het
                                                ;segment worden opgeslagen
                                                ;in locaties 100-103H.
0119                        start1:
0119    B8 0000             mov ax,0
011C    8E D8               mov ds,ax           ;data-segment naar 0000H
011E    BB 0100             mov bx,0100H        ;pointer naar int vector 100H
0121    83 7F 02 0          cmp word ptr [bx],0
0125    75 16               jne quit            ;programma reeds uitgevoerd, exit
0127    83 3F 00            cmp word ptr2 [bx],0
012A    75 11               jne quit            ;programma uitgevoerd, exit
012C    B8 0103 R           mov ax,offset usrprg
012F    89 07               mov [bx],ax         ;programma startadres
0131    8C C8               mov ax,cs
0133    89 47 02            mov [bx+2],ax       ;data-segment
0136    0E                  push cs
0137    1F                  pop ds
0138    BA 0141 R           mov dx,offset veryend
013B    CD 27               int 27h
013D                        quit:
013D    CD 20               int 20h
013F                        veryend:
013F                        double ends
                            end start
```

Voorbeeld 2 is een programma om de assembleerroutine in het opgegeven gebied van het geheugen te plaatsen.

Voorbeeld 2:

```
10 I=0:JC=0
100 DIM A%(23)
150 MEM%=VARPTR(A%(1))
200 FOR I=1 TO 23
300 READ JC
400 POKE MEM%,JC
450 MEM%=MEM%+1
500 NEXT
600 C1%=2:C2%=3:C3%=0
700 SOM=VARPTR(A%(1))
800 CALL SOM(C1%,C2%,C3%)
900 PRINT C3%
950 END
1000 DATA &H55,&H8b,&Hec,&H8b,&H76,&H08,&H8b,&H04,&H8b
1100 DATA &H76,&H0a,&H03,&H04,&H8b,&H7e,&H06,&H89,&H05
1200 DATA &H5d,&Hca,&H06,&H00
```

APPENDIX E
BASIC PROGRAMMA'S OMZETTEN IN GW-BASIC

Als een BASIC programma niet in GW-BASIC is geschreven, moet u enkele wijzigingen aanbrengen voordat u het kunt laten uitvoeren. In dit hoofdstuk wordt uitgelegd om welke wijzigingen het gaat.

TEKENREEKS-DIMENSIES

Wis alle opdrachten die worden gebruikt voor het opgeven van de tekenreekslengte. Een opdracht als:

```
DIM A$(I,J)
```

die de afmetingen van een tekenreeks-matrix opgeeft als J tekenreeksen met lengte I, moet worden omgezet in de onderstaande opdracht:

```
DIM A$(J)
```

Sommige BASIC-versies hanteren een komma of het en-teken (&) voor het koppelen van tekenreeksen. Voor GW-BASIC moeten deze tekens worden omgezet in het plusteken (+).

In GW-BASIC worden de functies MID$, RIGHT$ en LEFT$ gebruikt om een tekenreeks samen te stellen die bestaat uit een bepaald aantal tekens van een andere tekenreeks. Een functie als A$(I) om te verwijzen naar het teken op positie I in A$, of functies als A$(I,J) om een deeltekenreeks te maken met de tekens op de posities I tot J in A$, moeten als volgt worden omgezet:

andere BASIC	GW-BASIC
X$=A$(I)	X$=MID$(A$,I,1)
X$=A$(I,J)	X$=MID$(A$,I,J-I+1)

Als de verwijzing naar een deeltekenreeks links van het is-gelijk-teken staat en X$ wordt gebruikt voor het vervangen van tekens in A$, gaat het omzetten als volgt:

andere BASIC	GW-BASIC
A$(I)=X$	MID$(A$,I,1)=X$
A$(I,J)=X$	MID$(A$,I,J-I+1)=X$

MEERVOUDIGE INSTRUCTIES

In sommige versies van BASIC zijn opdrachten met de onderstaande opzet toegestaan om B en C gelijk te stellen aan nul:

```
10 LET B=C=0
```

GW-BASIC zou het tweede is-gelijk teken kunnen interpreteren als een logische operator en als gevolg daarvan B gelijk stellen aan 1 als C gelijk is aan 0. Gebruik daarom twee opdrachten:

```
10 C=0 : B=0
```

MEERVOUDIGE OPDRACHTEN

Sommige versies van BASIC maken gebruik van een backslash (\) om opdrachten op dezelfde, regel van elkaar te scheiden. Gebruik daarvoor bij GW-BASIC altijd de dubbele punt (:).

MAT-FUNCTIES

Programma's met MAT-functies moeten worden herschreven. Gebruik FOR...NEXT-constructies om te zorgen dat het programma op de juiste wijze wordt uitgevoerd.

FOR...NEXT

Sommige versies van BASIC voeren een FOR...NEXT-lus altijd minimaal één keer uit, ongeacht de waarde van de lusteller. GW-BASIC controleert eerst de tellerstand en voert de lus niet uit als de tellerstand buiten de begin- en eindstand in de FOR-opdracht ligt.

APPENDIX F
COMMUNICATIE

Deze appendix beschrijft de GW-BASIC opdrachten die nodig zijn voor asynchrone communicatie met andere computers en randapparatuur via de RS-232 poorten.

OPENEN VAN COMMUNICATIE-BESTANDEN

De OPEN COM-opdracht reserveert een buffer voor invoer en uitvoer, op dezelfde wijze als een OPEN-opdracht schijfbestanden opent.

I/O COMMUNICATIE

Omdat de communicatie-poort wordt geopend als een bestand, kunnen alle voor bestanden geldige I/O-opdrachten ook worden gebruikt voor COM.

De sequentiële invoeropdrachten van COM zijn hetzelfde als de invoer-opdrachten voor schijfbestanden:

INPUT#
LINE INPUT#
INPUT$

De sequentiële uitvoer-opdrachten van COM zijn hetzelfde als de uitvoer-opdrachten voor schijfbestanden:

PRINT#
PRINT# USING

Zie hoofdstuk 7 voor meer bijzonderheden.

COM I/O FUNCTIES

Het moeilijkste aspect van asynchrone communicatie is het snel genoeg verwerken van ontvangen informatie. Bij snelheden boven 2400 baud (bps) moet regelmatig een code naar de host worden gestuurd om het verzenden van gegevens tijdelijk te onderbreken omdat de invoerbuffer vol is. Zodra de in de buffer opgeslagen tekens zijn verwerkt, ontvangt de host de instructie het verzenden van gegevens te hervatten. U kunt overdracht van tekens tijdelijk onderbreken door XOFF (CTRL-S) naar de host te sturen. Met XON (CTRL-Q) kunt u de communicatie weer hervatten. U kunt deze commando's uiteraard alleen gebruiken als uw systeem hiervoor geschikt is.

GW-BASIC beschikt over drie functies waarmee u kunt bepalen of de capaciteit van de invoerbuffer wordt overschreden (ook wel "overrun" genoemd):

LOC(x) Geeft het aantal tekens in de invoerbuffer. Deze buffer kan meer dan 255 tekens bevatten (instelbaar via optie /c:). Als de buffer meer dan 255 tekens bevat, geeft LOC(x) toch de waarde 255. Aangezien de maximale tekenreeks-lengte ook 255 tekens is, is het niet noodzakelijk de tekenreeks-lengte te controleren voordat gegevens worden ingelezen.

LOF(x) Geeft aan hoeveel ruimte nog beschikbaar is in de invoerbuffer. Dit wordt als volgt aangegeven:

/c:(afmeting buffer)-aantal opgeslagen tekens

Met LOF kunt u controleren of de maximale opslagcapaciteit van de invoerbuffer is bereikt.

EOF(x) De status "waar" (-1) geeft aan dat de invoerbuffer leeg is en "onwaar" (0) dat de invoerbuffer nog tekens bevat.

FOUTMELDINGEN

De foutmelding **"Communicatiebuffer vol"** verschijnt als u probeert gegevens in te lezen terwijl de invoerbuffer vol is (dus als LOC(x) 0 is).

De foutmelding **"Invoer/Uitvoer fout"** verschijnt als een van de volgende fouten optreedt: overrun error (OE), framing error (FE) of break interrupt (BI). De fout wordt hersteld door volgende ontvangen tekens, maar het teken dat de fout veroorzaakte, gaat verloren.

De melding **"Onjuist apparaat"** verschijnt als tijdens invoer of uitvoer het signaal Data Set Ready (DSR) wegvalt.

De melding **"Pariteitfout"** verschijnt als in een OPEN COM-opdracht de optie PE is geactiveerd en tijdens de I/O een onjuiste pariteit wordt gedetecteerd.

DE INPUT$-FUNCTIE

Aangezien in communicatie alle ASCII-codes een betekenis hebben, is de INPUT$-functie beter geschikt voor het lezen van COM-bestanden dan de opdrachten INPUT en LINE INPUT. De opdracht INPUT onderbreekt de invoer bij elke komma of ENTER en is daarom het minst geschikt voor het lezen van COM-bestanden. LINE INPUT beëindigt de invoer bij een ENTER.

Bij INPUT$ kunnen alle gelezen tekens aan een tekenreeks worden toegekend.

INPUT$ geeft x tekens uit bestand y. De volgende opdrachten zijn het meest efficiënt bij het lezen van een COM-bestand:

```
10 WHILE NOT EOF(1)
20 A$=INPUT$(LOC(1),#1)
30 ...
40 ...Verwerk gegevens weergegeven in A$
50 ...
60 WEND
```

Met de bovenstaande opdrachten worden de volgende opdrachten gegeven: geef het aantal tekens in de invoerbuffer en sla deze tekens op in A$. Als de buffer meer dan 255 tekens bevat, worden maar 255 tekens per keer opgeslagen om een overflow in de tekenreeks te voorkomen. In dit geval is EOF(1) onwaar en wordt de invoervoortgezet totdat de invoerbuffer leeg is.

GET EN PUT OPDRACHTEN VOOR COM-BESTANDEN

Functie:
I/O voor COM-bestanden met een bepaalde lengte mogelijk maken.

Syntaxis:
GET *bestandsnummer*, *nbytes*
PUT *bestandsnummer*, *nbytes*

Opmerkingen:
Voor *bestandsnummer* moet u een geheel getal opgeven in het voor bestandsnummers geldige bereik.

Het argument *nbytes* is een geheel getal dat het aantal bytes weergeeft dat naar de bestandsbuffer moet worden gestuurd of uit de buffer moet worden opgehaald. U kunt de waarde die in de optie /s: voor *nbytes* is ingesteld niet overschrijden.

Het is niet aan te bevelen de opdrachten GET en PUT te gebruiken bij communicatie via telefoonlijnen, aangezien de kwaliteit daarvan nogal eens te wensen overlaat.

Voorbeeld:
Het onderstaande TTY-voorbeeldprogramma is bedoeld om te oefenen met I/O-communicatie. Met dit programma kunt u uw computer gebruiken als een conventionele terminal en is tevens full-duplex communicatie mogelijk. U kunt zowel ontvangen gegevens opslaan in een bestand als gegevens vanuit een bestand naar een ander systeem verzenden.

Dit programma toont niet alleen de aspecten van asynchrone communicatie, maar kan ook worden gebruikt voor het versturen van GW-BASIC programma's en gegevens vanuit of naar de computer.

Attentie:
Dit programma is geschreven voor communicatie met DEC® SYSTEM-20, met name met gebruik van XON en XOFF. Als u wilt communiceren met andere typen hardware kan het nodig zijn het programma enigszins te wijzigen.

VOORBEELDPROGRAMMA TTY

```
10 SCREEN 0,0:WIDTH 80
15 KEY OFF:CLS:CLOSE
20 DEFINT A-Z
25 LOCATE 25,1
30 PRINT STRING$(60," ")
40 FALSE=0:TRUE=NOT FALSE
50 MENU=5 'Waarde van MENU key (^E)
60 XOFF$=CHR$(19):XON$=CHR$(17)
100 LOCATE 25,1:PRINT "Async TTY programma";
110 LOCATE 1,1:LINE INPUT "Snelheid?";SNELHEID$
120 COMFIL$="COM1:,+SNELHEID$+",E,7"
130 OPEN COMFIL$ AS #1
140 OPEN "SCRN:"FOR OUTPUT AS #3
200 PAUSE=FALSE
210 A$=INKEY$:IF A$="" THEN 230
220 IF ASC(A$)=MENU THEN 300 ELSE PRINT #1,A$;
230 IF EOF(1) THEN 210
240 IF LOC(1)>128 THEN PAUSE=TRUE:PRINT #1,XOFF$;
250 A$=INPUT$(LOC(1),#1)
260 PRINT #3,A$;: IF LOC(1)>0  THEN 240
270 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
280 GOTO 210
300 LOCATE 1,1:PRINT STRING$(30,32):LOCATE 1,1
310 LINE INPUT "Bestand? ";DSKFIL$
400 LOCATE 1,1:PRINT STRING$(30,32):LOCATE 1,1
410 LINE INPUT"(T)ransmit of (R)eceive?";TXRX$
420 IF TXRX$="T" THEN OPEN DSKFIL$ FOR INPUT AS #2:GOTO 1000
430 OPEN DSKFIL$ FOR OUTPUT AS #2
440 PRINT #1,CHR$(13);
500 IF EOF(1) THEN GOSUB 600
510 IF LOC(1)>128 THE PAUSE=TRUE:PRINT #1,XOFF$;
520 A$=INPUT$(LOC(1),#1)
530 PRINT #2,A$;:IF LOC(1)>0 THEN 510
540 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
550 GOTO 500
600 FOR I=1 TO 5000
610 IF NOT EOF(1) THEN I=9999
620 NEXT I
630 IF I>9999 THEN RETURN
640 CLOSE #2;CLS:LOCATE 25,10:PRINT "* Download complete *";
650 RETURN 200
1000 WHILE NOT EOF(2)
1010 A$=INPUT$(1,#2)
1020 PRINT #1,A$;
1030 WEND
1040 PRINT #1,CHR$(28);^Z  om bestand te sluiten.
1050 CLOSE #2:CLS:LOCATE 25,10:PRINT "** Upload complete **";
1060 GOTO 200
9999 CLOSE:KEY ON
```

UITLEG BIJ VOORBEELDPROGRAMMA TTY

Opmerking: Bij asynchrone communicatie wordt de informatie als afzonderlijke tekens verstuurd in plaats van per regel of per blok. Daarom zijn alle PRINT-instructies in het voorbeeldprogramma (naar het COM-bestand of het beeldscherm) afgesloten door een puntkomma (;). Hierdoor wordt de line feed die gewoonlijk aan het einde van een PRINT-opdracht wordt gegeven, uitgesteld.

Opmerkingen bij de programmaregels:

10 Zet het scherm in de zwart/wit alfanumerieke modus en stelt de regellengte in op 80.

15 Verwijdert de informatie over de functietoetsen van schermregel 25, maakt het scherm leeg en zorgt dat alle bestanden worden gesloten.

20 Definieert alle variabelen als een geheel getal, vooral in verband met de subroutine op de regels 600-620. Het komt de verwerkingssnelheid ten goede als u voor de variabelenaam van de lusteller een integer-variabele kiest (indien mogelijk).

40 Definieert Booleaans "waar" en "onwaar".

50 Definieert de ASCII-code (ASC) van de menu-toets.

60 Definieert de ASCII-codes voor XON en XOFF.

100-130 Geeft de programmanaam weer en vraagt de om de baud rate. Opent het COM-bestand met bestandsnummer 1; hanteert even pariteit en 7 data bits.

200-280 In dit onderdeel wordt de full-duplex I/O tussen het beeldscherm en het apparaat dat aangesloten is op de RS-232-connector als volgt opgesteld:

1 Wacht op een ingetikt teken en sla het op in A$. INKEY$ geeft een lege tekenreeks tot een teken wordt ingetikt.

2 Als geen teken wordt ingetikt, blijf daar dan op wachten.

3 Zodra een teken werd ingetikt:

- Als het teken de MENU-toets is, dan wil de gebruiker een bestand laden. Ga naar regel 300.
- Als het teken in A$ niet de MENU-toets is, stuur het dan naar het communicatie-bestand (PRINT #1...).

4 Regel 230: controleer of de COM-buffer tekens bevat. Als dat niet zo is, verlaat dan de buffer en controleer het toetsenbord.

5 Regel 240: zijn meer dan 128 tekens aanwezig, geef dan de instructie tot een PAUSE om het verzenden van gegevens te onderbreken en verzendt de code voor XOFF$ naar de host.

6 Regels 250-260: lees de inhoud van de COM-buffer en geef deze op het scherm weer totdat de buffer leeg is. Blijf het aantal tekens in de buffer controleren. Hervat de gegevensoverdracht pas als de buffer geen tekens meer bevat.

7 Hervat de overdracht door XON naar de host te versturen. Dit kan alleen als de verzending door XOFF werd onderbroken.

8 Herhaal dit proces totdat de MENU-toets wordt ingedrukt.

300-310 Vraag om het bestand waarin de gegevens moeten worden opgeslagen.

400-420 Vraag of het genoemde bestand moet worden verzonden (upload, Transmit) of ontvangen (down-load, Receive) en open het bestand met bestandsnummer 2 voor uitvoer, resp. invoer.

430 "Receive"-routine. Stuurt de code voor ENTER naar de host om de gegevensoverdracht te laten beginnen. Dit programma neemt aan dat dit commando al naar de host was gestuurd, maar nog niet was bevestigd met ENTER. Als de host een DEC-systeem is en op de MENU-toets werd gedrukt in plaats van op ENTER, zou het commando als volgt kunnen zijn:

COPY TTY:=MANUAL.MEM (MENU-toets)

500 Als geen tekens meer worden ontvangen, (LOC(x) wordt 0), dan gaat het programma naar een timeout-routine.

510 Als de buffer meer dan 128 tekens bevat, geef dan instructie voor een pauze op en stuur XOFF naar de host.

520-530 Lees alle tekens in de COM-buffer (LOC(x)) en schrijf deze weg op diskette (PRINT #2...) tot de buffer leeg is.

540-550 Als opdracht is gegeven voor een pauze, beëindig dan de pauze en geef dan de host opdracht het verzenden van gegevens te hervatten, door XON te versturen. Herhaal het proces tot gedurende een bepaalde tijd geen gegevens meer worden ontvangen.

600-650 Time-out subroutine. De waarden in de FOR-opdracht werden bepaald door een aantal verschillende waarden uit te testen. Als gedurende 17-20 seconden geen teken wordt ontvangen, wordt aangenomen dat de overdracht is voltooid. Als in deze tijd wel een teken wordt ontvangen (regel 610) kies dan voor I een waarde die veel hoger is. Sluit het schijfbestand en ga terug naar regel 200.

1000-1060 "Transmit"-routine. Sla met behulp van de INPUT$-opdracht een teken op in A$, tot het einde van het schijfbestand is bereikt. Stuur een teken naar de COM-buffer (regel 1020). Regel 1040: verstuur ^Z bij het bereiken van het einde van het bestand, indien het aangesloten apparaat deze code nodig heeft voor het sluiten van het bestand. Regels 1050 en 1060: sluit schijfbestand, meldt dat de gegevensoverdracht is voltooid en ga naar regel 200.

9999 Deze regel wordt in het voorbeeldprogramma nooit uitgevoerd. Voeg als oefening enkele regels toe aan routine 400-420 om het programma via regel 9999 te verlaten. Deze regel sluit de nog niet gesloten COM-bestand en laat de functietoetsen weer op het scherm verschijnen.

APPENDIX G
EQUIVALENTEN VAN HEXADECIMALE WAARDEN

Onderstaande tabel geeft een overzicht van de decimale en binaire equivalenten van hexadecimale waarden.

Decimale en binaire equivalenten van hexadecimale waarden

hexadecimaal	decimaal	binair
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Onderstaande tabel geeft een overzicht van de decimale equivalenten van hexadecimale waarden.

hexade-cimaal	decimaal	hexade-cimaal	decimaal
0	0	80	128
1	1	.	
2	2	.	
3	3	.	
4	4	90	144
5	5	.	
6	6	.	
7	7	.	
8	8	A0	160
9	9	.	
A	10	.	
B	11	.	
C	12	B0	176
D	13	.	
E	14	.	
F	15	.	
10	16	C0	192
11	17	.	
12	18	.	
13	19	.	
14	20	D0	208
15	21	.	
16	22	.	
17	23	.	
18	24	E0	224
19	25	.	
1A	26	.	
1B	27	.	
1C	28	F0	240
1D	29	100	256
1E	30	200	512
1F	31	300	768
20	32	400	1024
.	.	500	1280
.	.	600	1536
.	.	700	1792
30	48	800	2048
.	.	900	2304
.	.	A00	2560
.	.	B00	2816
40	64	C00	3072
.	.	D00	3328
.	.	E00	3584
.	.	F00	3840
50	80	1000	4096
.	.	2000	8192
.	.	3000	12288
.	.	4000	16384
60	96	5000	20480
.	.	6000	24576
.	.	7000	28672
.	.	8000	32768
70	112	9000	36864
.	.	A000	40960
.	.	B000	45056
.	.	C000	49152
		D000	53248
		E000	57344
		F000	61440

APPENDIX H
SCAN-CODES VOOR TOETSEN

Toets	scan-code
ESC	01
1/!	02
2/@	03
3/#	04
4/$	05
5/%	06
6/^	07
7/&	08
8/*	09
9/(	0A
0/)	0B
-/_	0C
=/+	0D
BACKSPACE	0E
TAB	0F
Q	10
W	11
E	12
R	13
T	14
Y	15
U	16
I	17
O	18
P	19
[/{	1A
]/}	1B
ENTER	1C
CTRL	1D
A	1E
S	1F
D	20
F	21
G	22
H	23
J	24
K	25
L	26
;/:	27
'/"	28
`/~	29
linker-SHIFT	2A
\/|	2B
Z	2C
X	2D
C	2E

Toets	scan-code
V	2F
B	30
N	31
M	32
,/<	33
//?	35
rechter-SHIFT	36
*/PRTSC	37
ALT	38
SPATIEBALK	39
CAPS LOCK	3A
F1	3B
F2	3C
F3	3D
F4	3E
F5	3F
F6	40
F7	41
F8	42
F9	43
F10	44
NUM LOCK	45
SCROLL LOCK	46
7/HOME	47
8/CURSOR OMHOOG	48
9/PGUP	49
-	4A
4/CURSOR LINKS	4B
5	4C
6/CURSOR RECHTS	4D
+	4E
1/END	4F
2/CURSOR OMLAAG	50
3/PGDN	51
0/INS	52
./DEL	53

APPENDIX I
DOOR GW-BASIC HERKENDE TEKENS

De tekenset van GW-BASIC bevat alle tekens die mogen worden gebruikt in commando's, opdrachten, functies en variabelen die in deze handleiding zijn beschreven. De set bestaat uit het alfabet, cijfers en speciale tekens. GW-BASIC maakt geen onderscheid tussen hoofdletters en kleine letters.

In de onderstaande tabel wordt een overzicht gegeven van de speciale tekens en toetsen die door GW-BASIC worden herkend:

teken	omschrijving
	spatie
=	is-gelijk teken, ook voor toekennen van waarden
+	plusteken, ook voor koppelen van tekenreeksen
-	minteken
*	asterisk, vermenigvuldigteken
/	slash, deelteken
^	caret, machtsverheffen, verwijst ook naar <CTRL>
(	open-haak
)	sluit-haak
%	percentteken, ook aanduiding voor geheel getal
#	aanduiding voor getal en voor getal met dubbele precisie
$	dollar-teken, ook aanduiding voor tekenreeks
!	uitroepteken, ook aanduiding voor getal met enkele precisie
[	vierkant haakje open
]	vierkant haakje sluiten
,	komma, ook scheidingsteken
"	dubbel aanhalingsteken, ook scheidingsteken voor tekenreeksen
.	punt of decimale punt
'	enkel aanhalingsteken, apostrof of aanduiding voor opmerking (i.p.v. REM)
;	puntkomma, ook onderdrukking van carriage return
:	dubbele punt, scheidingsteken tussen opdrachten
&	en-teken, ook aanduiding voor hexadecimale en octale getallen
?	vraagteken
<	kleiner dan-symbool
>	groter dan-symbool
\	backslash, ook symbool voor integer-delen
@	"apestaart"
_	onderstreepteken
BACKSPACE	wist het teken vóór de cursor
ESC	wist de huidige regel van het beeldscherm
TAB	verplaatst de print-kop of cursor naar de volgende tabulatorstop (na elke acht tekens bevindt zich een tabulatorstop)
CURSOR	verplaatst de cursor naar de volgende fysieke regel
ENTER	sluit een regel of instructie af en verplaatst de cursor naar het begin van de volgende regel, of voert de opdracht uit in de commando-modus.

WOORDENLIJST

A

abend
Acroniem van "abnormal end of task". Een abend is de voortijdige beëindiging van een taak waaraan de computer bezig was. Een dergelijke onderbreking wordt meestal veroorzaakt door een fout die niet kan worden opgelost met de geprogrammeerde hulpmiddelen voor "recovery".

Acroniem
Een woord dat bestaat uit de eerste letters van woorden, of uit de eerste letters plus delen van verschillende woorden. In computertechnologie wordt veelvuldig gebruikt gemaakt van acroniemen. Bijvoorbeeld: BASIC is het acroniem voor "Beginners All-purpose Instruction Code" en COBOL is het acroniem voor "Common Business Oriented Language".

Actieve partitie
Deel van het computergeheugen waarin het besturingssysteem is opgeslagen.

Adres
Een naam, label of nummer ter identificatie van een register of locatie waarin informatie is opgeslagen.

Alfabetisch
Het alfabetisch ordenen van gegevens, in tegenstelling tot sorteren op basis van numerieke volgorde.

Alfanumeriek
Een combinatie van de woorden "alfabetisch" en "numeriek". Een combinatie van letters en cijfers.

Algebraïsche taal
Een taal waarvan de opdrachten zo zijn opgesteld dat ze de structuur van een algebraïsche uitdrukking benaderen. Een goed voorbeeld van een algebraïsche taal is Fortran.

Algoritme
Een aantal goed gedefinieerde regels en procedures die in een bepaalde volgorde moeten worden uitgevoerd om de oplossing van het probleem in een bepaald aantal stappen te kunnen verkrijgen. In een algoritme kunnen procedures en instructies van allerlei typen worden gebruikt waaronder rekenkundige, algebraïsche en logische. Alhoewel algoritmen eenvoudig of complex kunnen zijn, moet de oplossing binnen een bepaald aantal stappen worden bereikt. Als u een computer gebruikt voor het oplossen van problemen is het gebruik van algoritmen noodzakelijk. Computers kunnen problemen namelijk alleen analyseren en oplossen als die zijn opgesteld in de vorm van instructies die voldoen aan vastliggende conventies.

Apparaat
Hardware die een bepaalde functie kan verrichten, zoals diskette-stations, printers.

Argument
1. Een type variabele waarvan de waarde niet direct een functie is van een andere variabele. In een argument kan de waarde van een getal of een variabele worden opgeslagen.

2. Een getal dat wordt opgegeven bij een voorgeprogrammeerde functie. Voorbeeld: SQR(X) berekent de vierkantswortel van het getal X.

Array
Zie *Matrix*.

ASCII
Acroniem van American Standard Code for Information Interchange, een gestandaardiseerde 8-bits code. Het omzetten van tekens in hun ASCII-codes wordt vaak gebruikt bij uitwisseling van informatie tussen verschillende systemen. ASCII is ontwikkeld door het American National Standards Institute (ANSI). Van elke 8 bits worden er 7 gebruikt voor gegevens, de achtste bit wordt gebruikt voor de pariteit.

Assembleerprogramma
Een computerprogramma voor het opstellen van assembleerroutines die direct kunnen worden uitgevoerd door de computer.

Assembler (assembleertaal)
Een symbolische programmeertaal die meer georiënteerd is op elke afzonderlijke processor, dan op het schrijven van toepassingsprogramma's. Een in assembleertaal geschreven routine wordt door een assembleerprogramma omgezet in machinetaal. Symbolen die verwijzen naar geheugenlocaties worden omgezet in numerieke geheugenlocaties. Symbolische bewerkingscodes worden omgezet in numerieke bewerkingscodes.

Asynchroon
1. Bewerkingen waarvan de uitvoering niet wordt gestuurd door de pulsen van de interne klok in een computersysteem.

2. Een soort bewerking waarbij aan een nieuwe instructie wordt begonnen als de voorafgaande is voltooid. Voor het uitvoeren van de instructies bestaat dus geen vast tijdschema. De huidige instructie moet eerst worden afgerond voordat aan de volgende kan worden begonnen, ongeacht hoe lang de uitvoering in beslag neemt.

Asynchrone communicatie
Seriële gegevensoverdracht tussen apparaten waarbij de overdracht van elk teken wordt voorafgegaan door een startbit en gevolgd door een stopbit. Deze wijze van overdracht wordt ook wel start/stop-overdracht genoemd.

B

Backup
1. Een kopie van gegevens op een diskette of een ander medium waardoor de gegevens op het originele medium beveiligd zijn tegen verlies of beschadiging.

2. Reserve-apparatuur die kan worden ingeschakeld als de gebruikte apparatuur defect raakt.

BASIC
Acroniem van Beginner's All-purpose Symbolic Instruction Code. Deze computer programmeertaal is ontwikkeld op het Dartmouth College als een hulpmiddel bij het leren van de fundamentele programmeertechnieken. BASIC heeft sindsdien bekendheid verworven als een van de eenvoudigste programmeertalen.

Batch processing (stapelgewijze verwerking)
Het zodanig programmeren van de computer dat een enkel programma of een aantal programma's van hetzelfde type moet worden afgerond voordat aan het volgende type programma kan worden begonnen.

Baud
Een eenheid voor het uitdrukken van transmissiesnelheid. Baud verwijst naar het aantal bits dat per seconde wordt verstuurd. Aangezien een teken uit meerdere bits kan bestaan, is het niet zo dat baud per definitie verwijst naar het aantal tekens per seconde. Gebruikelijke baud rates zijn 110, 300, 1200, 2400, 4800 en 9600.

Bestand
Een verzameling gegevens die met elkaar in verband staan of een verzameling programma's die door de computer als een eenheid worden beschouwd.

Bestand-bescherming
Methoden of procedures die verhinderen dat gegevens op een opslagmedium of in een bestand per ongeluk worden gewist.

Bestandsnaam
De unieke naam die, meestal door de gebruiker, wordt toegekend aan een bestand om deze bij volgende bewerkingen te kunnen herkennen en terugvinden.

Bestandsstructuur
Concept van de samenhang tussen gegevens, waarden, records en bestanden. De structuur geeft meestal aan hoe gegevens worden opgeslagen en verwerkt.

Bestandstype
Een set die bestaat uit minimaal 1 en maximaal 3 tekens en wordt geplaatst achter de bestandsnaam. Het bestandstype is een nadere bepaling of verduidelijking bij de bestandsnaam. Tussen de bestandsnaam en een bestandstype moet altijd een punt worden geplaatst. Voorbeeld: HELGA.BAS.

Besturingssysteem
Wordt ook vaak "operating system" genoemd. Het stelt de computer in staat toepassingsprogramma's uit te voeren en bestuurt alle basisfuncties van de computer.

Binair
1. Een kenmerk dat of een eigenschap die slechts twee toestanden kan aannemen, bijvoorbeeld 0 of 1, "waar" of "onwaar".

2. Een getallenstelsel waarbij wordt uitgegaan van het grondtal 2 in plaats van 10, zoals in het decimale stelsel. In het binaire stelsel wordt slechts gebruik gemaakt van twee cijfers, 0 en 1.

3.Een binair apparaat, zoals een computer, hanteert bij de verwerking van gegevens twee toestanden of niveau's.

Binair getal
Een getal uitgedrukt als een combinatie van de cijfers 0 en 1.

Bit
Samentrekking van "binary digit". Een bit kan 1 of 0 zijn en is de kleinste informatie-eenheid die door de computer wordt herkend.

Blok
Een hoeveelheid geheugenruimte of gegevens, van willekeurige lengte, meestal aaneengesloten en met een interne samenhang. Een blok bestaat meestal uit meerdere soortgelijke records die samen als een eenheid worden beschouwd.

Booleaanse logica
Een onderdeel van de wiskunde dat vooral wordt gebruikt voor het opstellen van vergelijkingen. In een instructie kan de inhoud van twee velden of variabelen worden vergeleken. Op basis van het resultaat van die vergelijking kan de inhoud van een of meer (andere) variabelen worden gewijzigd. Dergelijke vergelijkingen worden ook vaak toegepast om voorwaarden te testen en het programmaverloop te besturen. De Booleaanse logica is ontwikkeld door de Britse wiskundige George Boole (1815-1864). Voorbeelden van booleaanse operatoren zijn OR, AND, NOT, XOR, EQV, en IMP.

Boot
Het opstarten van een computersysteem wordt "booting" genoemd. Bij het opstarten wordt het gehele systeem geïnitialiseerd en het besturingssysteem (operating system) geactiveerd. Het woord "boot" is een afkorting van het woord "bootstrap".

Bps
Bits per seconde.

Buffer
Een geheugen waarin gegevens tijdelijk worden opgeslagen of van waaruit gegevens worden verstuurd naar verschillende apparaten.

Byte
Een gegevenselement dat bestaat uit 8 bits plus een pariteits-bit. In 1 byte kan een letter of speciaal teken worden opgeslagen, maar ook twee cijfers of acht binaire bits. Een byte wordt meestal gecodeerd naar ASCII.

C

Central processor unit (CPU)
In het Nederlands vaak aangeduid als "centrale verwerkingseenheid". Het hart van het computersysteem waar gegevens worden verwerkt en berekeningen worden uitgevoerd. De CPU bevat een controle-eenheid voor het interpreteren en uitvoeren van instructies en een rekeneenheid voor het uitvoeren van berekeningen en logische bewerkingen. De CPU zorgt ook voor het interne transport van informatie, regelt de in- en de uitvoer van gegevens en kan informatie tijdelijk opslaan.

COBOL
Acroniem van "Common Business Oriented Language". Deze computertaal is geschikt voor het schrijven van complexe zakelijke toepassingsprogramma's. COBOL is ontwikkeld door CODASYL, een comité bestaande uit vertegenwoordigers van het Amerikaanse Ministerie van Defensie, een aantal computerfabrikanten en de belangrijkste afnemers van gegevensverwerkende apparatuur. COBOL is speciaal ontwikkeld voor het manipuleren en verwerken van gegevens. COBOL hanteert een aantal nauwkeurige en gestandaardiseerde regels.

Coderen
1. Instructies schrijven voor een computersysteem.
2. Gegevens in willekeurige tabellen rangschikken.
3. Gebruik maken van machinetaal.
4. Programmeren.

Commando
Pulsen, signalen, woorden of letters die de computer opdracht geven te stoppen, te beginnen of verder te gaan met een bewerking of in een instructie. Het woord "commando" wordt vaak ten onrechte gebruikt als synoniem voor het woord "instructie".

Compatibel
Wordt gezegd van gegevens, programma's of apparatuur, geschikt voor meerdere types computers of apparatuur.

Compiler
Een computerprogramma geschikt om een programma, geschreven in een programmeertaal, te vertalen naar de machinetaal die door de betreffende computer wordt gebruikt.

Computer-netwerk
Een verspreide configuratie van computerapparatuur, aan elkaar gekoppeld via communicatielijnen. Dankzij een dergelijk netwerk kan elk systeem binnen het netwerk elders aanwezige programmatuur benutten of gegevens verwerken, of bijvoorbeeld gebruik maken van elders opgestelde printers of opslagmedia.

Configuratie
In hardware: een aantal apparaten die onderling met elkaar verbonden zijn en samen één systeem vormen.
In software: het totaal aan software-modules en de bijbehorende onderlinge relaties.
Configuratie verwijst echter ook vaak naar een computersysteem inclusief de randapparatuur en de programma's.

Constante
Een waarde die niet verandert.

Co-processor
Een aan de centrale microprocessor gekoppelde microprocessor, geschikt voor het uitvoeren van speciale berekeningen (zoals berekeningen met decimale getallen). Een systeem dat is uitgerust met een mathematische co-processor zal berekeningen vaak sneller doorvoeren dan een systeem dat niet over een dergelijke co-processor beschikt.

Cursor
Een (knipperend) streepje of blokje op het beeldscherm dat de positie aangeeft waar het eerstvolgende ingetikte teken zal verschijnen.

D

Debug
Een Engelse term voor het opsporen en corrigeren van fouten in een programma of in apparatuur.

Decimale getallen.
Bij het gebruik van decimale getallen zorgt de computer automatisch dat de decimale punt altijd op de juiste positie in het getal komt te staan.

Decimale routine
Een aantal programma-instructies waardoor een computer die een decimale punt niet automatisch op de juiste wijze verwerkt toch bewerkingen met decimale getallen kan uitvoeren.

Detailbestand
Een gegevensbestand dat bestaat uit records met dezelfde kenmerken, maar die gegevens bevatten die van nature kunnen wijzigen zoals gegevens in een loonadministratie. Vergelijk met "master-bestand".

Diagnostische programma's
Programma's die speciaal zijn ontworpen voor het instellen en testen van apparatuur of het opsporen van fouten.

Directory
Engelse term voor "inhoudsopgave". Een tabel met daarin de naam, locatie en grootte van elk bestand dat op een geheugenmedium is opgeslagen. Tevens wordt in de directory aangegeven op welke datum het bestand is aangemaakt of voor het laatst is opgeslagen.

Directory met boomstructuur
Bestandstructuur bestaande uit directories en subdirectories, die op de vertakking van een boom lijkt.

Diskette
Een platte, flexibele schijf bedekt met magnetisch materiaal, in een beschermhoesje, die wordt gebruikt voor het opslaan van software en gegevens.

Diskette-station
Een apparaat waarin diskettes worden geplaatst voor het opslaan of inlezen van gegevens.

Disk Operating System
DOS. Een verzameling procedures en technieken waardoor de computer kan communiceren met een of meer stations voor het invoeren en opslaan van gegevens. Meestal wordt gebruik gemaakt van de afkorting DOS.

DOS
Acroniem voor Disk Operating System.

Double-density (Dubbele dichtheid)
Een type diskette met een opslagcapaciteit die ongeveer twee maal zo groot is als die van de standaard single-density diskettes.

Double-sided (Dubbelzijdig)
Bij een double-sided diskette zijn beide zijden geschikt voor het opslaan van gegevens.

Drive
Zie *Diskette-station.*

Dubbele precisie
Door variabelen te kenmerken als variabelen met dubbele precisie, wordt een hogere rekennauwkeurigheid gerealiseerd. Het is ook mogelijk de dubbele precisie in te schakelen via de optie /d op de commandoregel van GW-BASIC. Bij een getal met dubbele precisie rekent de computer intern met veel meer cijfers achter de decimale punt. Voor elk getal met dubbele precisie gebruikt het systeem twee bytes, zodat het rekenen met dubbele precisie meer geheugencapaciteit vergt.

E

End-Of-File teken (EOF)
Een teken dat aanduidt dat het laatste record van een bestand is gelezen.

Enkele precisie
De standaard precisie bij getallen. Bij getallen met enkele precisie rekent de computer met minder decimalen dan bij dubbele precisie. Het voordeel is echter dat enkele precisie minder geheugenruimte vergt. Voor elk getal met enkele precisie gebruikt het systeem één byte.

Exponent
Een symbool rechtsboven een variabelenaam of een getal dat aangeeft tot welke macht het getal moet worden verheven. Bijvoorbeeld: in A^2 is 2 de exponent. A^2 betekent A maal A.

F

File
Zie *Bestand*.

Fixed disk
Zie *Vaste schijf*.

Format
Vooraf vastgestelde layout of presentatiewijze van gegevens voor de opslag in een extern geheugenmedium, of de weergave via het beeldscherm of de printer.

Foutmelding
Melding op het beeldscherm of geluidssignaal waarmee wordt aangeduid dat er een fout is opgetreden in de hardware of software, dat gegevens op onjuiste wijze werden ingevoerd, of dat een instructie niet is toegestaan.

Functie
Een bewerking die is opgeslagen als voorgeprogrammeerde functie. Voorbeelden in GW-BASIC: COS, EOF, INSTR, LEFT$, SQR en TAN.

Functietoetsen
Voorgeprogrammeerde toetsen die de computer een bepaalde opdracht geven. De functie van de toetsen is niet binnen elk toepassingsprogramma identiek.

G

Gegeven
Algemene term (Engels: data) voor het aanduiden van informatie die door de computer kan worden verwerkt. Zie *Informatie*. De term verwijst ook naar één gegevenseenheid, zoals een getal.

Gegevensbestand (data file)
Een verzameling records die met elkaar in verband staan en op een bepaalde manier zijn gerangschikt. Gegevensbestanden bevatten computer-records met informatie waarop door de computer bewerkingen moeten worden uitgevoerd.

Geheel getal (integer)
Een getal zonder decimalen. Het gehele of natuurlijke getal. Bijvoorbeeld: 65 is een geheel getal, 65.1 niet en 65. ook niet.

Geheugen
Dat deel van de computer waar gegevens worden opgeslagen.

Geïntegreerde schakeling
Een complete elektronische schakeling, bestaande uit halfgeleider-substraten, ondergebracht in een miniatuur behuizing.

GIGO
Niet-officiële term voor slordig programmeerwerk en daaruit vaak resulterende onnauwkeurige verwerking van informatie. Acroniem van Garbage In Garbage Out. GIGO wordt meestal gebruikt om duidelijk te maken dat als gegevens op onjuiste wijze worden ingevoerd (garbage in) ze ook slecht worden weergegeven (garbage out).

Globaal zoeken
Globaal zoeken verwijst naar het zoeken van een term, variabele, commando, opdracht, e.d. Bij globaal zoeken zoekt de computer alle gevallen op waarin het gezochte gegeven voorkomt.

Grafische weergave
Hard- en of software die het mogelijk maakt informatie grafisch weer te geven, bijvoorbeeld op een voor grafische patronen geschikt beeldscherm of via printer of plotter. Vaak is ook interactie mogelijk via graphics tablets en een lichtpen.

Graphics
Zie *Grafische weergave.*

Grondtal
Het basisgetal van een getallenstelsel. Zo is 10 het grondtal, of de basis, van het meest gebruikte logaritmestelsel en ook van het decimale getallenstelsel.

H

Hard copy
Via de printer afgedrukte informatie.

Hard disk
Zie *Vaste schijf.*

Hardware
De apparatuur van een computersysteem, inclusief diskette-stations, printers e.d.

Hexadecimaal stelsel
Een getalstelsel dat is gebaseerd op het grondtal 16. In dit stelsel worden de cijfers 0 t/m 9 gebruikt en de letters A, B, C, D, E en F voor de hexadecimale equivalenten van de decimale getallen 10 t/m 15. In dit stelsel bestaan alle getallen uit machten van het grondtal 16. Voorbeeld: decimaal 22 is hexadecimaal 16, namelijk: $1x16^1 + 6x16^0$.

Huishoudelijke functies
Deze term verwijst naar diverse routine-handelingen die moeten worden uitgevoerd voordat de gegevensverwerking kan worden begonnen, of nadat de verwerking is voltooid. Voorbeelden: regelmatig backups maken, directories bijwerken e.d.

I

Informatie
Uit gegevens afgeleide resultaten. De computer verwerkt en produceert gegevens. De betekenis die uit die gegevens wordt afgeleid heet informatie; met andere woorden informatie resulteert uit gegevens. Deze twee woorden zijn geen synoniemen, maar worden wel vaak verwisseld.

Ingebouwde klok
Een echte klok waardoor uw programma in staat is gebruik te maken van de juiste tijd en datum. Met een in MS-DOS ingebouwde klok kunt u nagaan hoe lang het duurt voor een programma in zijn geheel is uitgevoerd. De klok houdt de tijd bij, ook als de computer is uitgeschakeld. De klok kan ook worden gebruikt voor het bijhouden van een persoonlijke kalender.

Inkorten
Beëindiging van een (reken)bewerking volgens bepaalde regels. Bijvoorbeeld het negeren van de cijfers achter de decimale punt in plaats van afronden, of het weglaten van tekens aan het einde van een regel bij het kopiëren van een bestand.

Input
Zie *Invoer*.

Input/output
Een algemene term die wordt gebruikt voor communicatie tussen de computer en randapparatuur. Meestal wordt de afkorting I/O gebruikt.

Instructie
Een stap van een programma waarin aan de computer duidelijk wordt gemaakt welke handeling hij moet uitvoeren. Het woord "instructie" wordt vaak ten onrechte gebruikt als synoniem voor het woord "commando". Een instructie kan uit meerdere commando's, opdrachten en variabelen bestaan.

Interface
Een schakeling of programma voor het uitwisselen van informatie tussen computers en randapparatuur, (zoals printers, monitors of modems) of tussen twee of meer computers (bijvoorbeeld in een netwerk).

Interpreter
Een interpreter leest een gebruikersprogramma, bijvoorbeeld een BASIC programma. Dit programma wordt regel voor regel gelezen, vertaald en uitgevoerd. Dit in tegenstelling tot een compiler die het gehele gebruikersprogramma eerst leest en daarna pas uitvoert. Een interpreter geeft dan ook pas een foutmelding als hij bij de regel aankomt waar de fout staat. Een compiler test het gehele programma en geeft onmiddellijk aan dat ergens een fout staat.

Invoer
1. Het proces voor het invoeren van gegevens in een computer.

2. De gegevens die worden ingevoerd in een computer.

I/O
Acroniem van Input/Output

K

K
Symbool voor de hoeveelheid 2^{10}, wat gelijk is aan 1024. De term "Kbyte" verwijst naar kilo-byte.

Koppelen
Het samenvoegen van groepen gegevens, zoals bestanden, in één set, bijvoorbeeld een nieuw bestand. In het Engels wordt hiervoor vaak de term "concatenate" gebruikt, die letterlijk "aaneenschakelen" betekent. Een gekoppelde gegevensset is een verzameling van gegevenssets die logisch met elkaar zijn verbonden.

L

Lees/schrijf-operaties
Technieken en programma's die worden gebruikt voor de overdracht van gegevens tussen het werkgeheugen en andere onderdelen van het computersysteem.

Logaritme
De logaritme L van getal X geeft aan tot welke macht E het grondtal G moet worden verheven om het oorspronkelijke getal X te verkrijgen. Dit betekent $G^E=X$. Voorbeeld: Stel G=10, X=1000. Dan geldt dus: omdat $10^3=1000$, is E=3 en is 3 de 10-logaritme van 1000.

M

M
Symbool voor aanduiding van 1 miljoen (10^6). Als dit symbool wordt gebruikt om geheugencapaciteit aan te geven, komt het overeen met 1.048.576 (220).

Mantisse
De mantisse van een getal zijn de cijfers voor de decimale punt. Bijvoorbeeld: bij het getal 24,2 is "24" de mantisse is en "2" de exponent. Het decimale getal wordt gelezen als .24 x 102.

Master-bestand
Een gegevensbestand dat bestaat uit records met kenmerken die zelden worden gewijzigd. Een goed voorbeeld van een master-bestand is een bestand met de naam en het adres van een employé, zijn datum van indiensttreding, zijn salarisnummer e.d.

Matrix
Een geordende verzameling gegevens. Via een bepaalde waarde voor het argument achter de matrix-naam wordt naar elk afzonderlijk gegeven verwezen. Voorbeeld: A(23) verwijst naar de inhoud van cel 23 in matrix A.

Menu
Een lijst met alternatieven waaruit de gebruiker een taak of handeling kan kiezen die door computer moet worden uitgevoerd.

Microprocessor
Een "central processing unit" (CPU), bestaande uit geïntegreerde schakelingen.

Modem
Acroniem van modulator demodulator. Een modem zet gegevens van de computer om in analoge signalen die kunnen worden getransporteerd via telefoonlijnen, of zet de signalen van de telefoonlijn om in een voor de computer bruikbare vorm.

MS-DOS
Acroniem van Microsoft Disk Operating System.

N

Nesten
Het is toegestaan routines of programmalussen te nesten. Dit houdt in dat een routine of lus binnen een andere routine of lus wordt opgenomen.

Nul-string
Een nul-string heeft een lengte van 0 tekens, d.w.z. bevat geen enkele informatie. Een dergelijke tekenreeks wordt ook wel "lege tekenreeks" genoemd. Dit is niet hetzelfde als een tekenreeks waarin bijvoorbeeld alleen een 0 of een spatie staat, want dat is wel zinnige informatie.

Numeriek
Numerieke gegevens zijn gegevens waarmee de computer kan rekenen, dus gegevens die als getallen worden beschouwd; dit in tegenstelling tot letters en andere tekens.

O

Octaal stelsel
Een getalstelsel dat is gebaseerd op het grondtal 8. In dit stelsel worden alleen de cijfers 0 t/m 8 gebruikt. In dit stelsel bestaan alle getallen uit machten van het grondtal 8. Voorbeeld: het decimale getal 15 is octaal 17, namelijk $1x8^1 + 7x8^0$. Computers maken gebruik van het octale stelsel om op relatief eenvoudige wijze binaire getallen te kunnen aangeven.

Opdracht
Een opdracht kan deel uitmaken van een instructie, eventueel gecombineerd met commando's, operatoren e.d. Een opdracht maakt deel uit van een reeks standaardopdrachten van de gebruikte programmeertaal.

Operand
Een operand is een gegeven waarop een bewerking wordt uitgevoerd. Een operand wordt meestal opgegeven via het adresgedeelte van een instructie, maar kan ook het resultaat, een parameter of een aanduiding zijn van de naam of locatie van de volgende instructie.

Operator
Een symbool dat aanduidt welk type bewerking wordt uitgevoerd; de operator zelf is het onderwerp van de bewerking. Er zijn wiskundige, relationele, logische en functionele operatoren.

Opslagmedium
Het magnetische materiaal waarop gegevens worden opgenomen en opgeslagen. Bijvoorbeeld: banden, diskettes, cassettes e.d.

Optie
Op de commandoregel van GW-BASIC kunnen een aantal opties worden ingesteld, waardoor het systeem een andere dan de standaardinstelling hanteert (bijvoorbeeld dubbele precisie in plaats van enkele precisie).

Optioneel argument
Een argument dat in een instructie mag worden weggelaten.

Output
Zie *Uitvoer*.

P

Parallelle uitvoer
Methode waarbij alle bits van een binair woord (byte) gelijktijdig worden verstuurd.

Parameter
Een variabele waaraan binnen een programma of routine een bepaalde waarde wordt toegekend. Een instelbaar kenmerk van gegeven, apparaat of systeem.

Pariteit
Een extra bit in een binair woord, die wordt gebruikt voor het opsporen van fouten in het geheugen of tijdens het uitwisselen van gegevens met randapparatuur. Door deze extra bit wordt getest of de som van een byte oneven of even is.

Partitie
Een bepaald deel van de vaste schijf, dat gereserveerd is voor een speciaal doel, bijvoorbeeld voor het besturingssysteem.

Pixel
Acroniem van "picture element". Een pixel is één beeldpunt op het scherm, waarnaar via één bit kan worden verwezen.

Pointer
Een pointer verwijst naar het adres van een ander gegeven of naar het adres van een ander record dat logisch met het eerste is verbonden.

Poort
Een in- of uitgangskanaal voor communicatie tussen de computer en randapparaten.

Programma
Een aantal genummerde programmaregels, met instructies die voldoen aan de conventies van een bepaalde programmeertaal. Via deze instructies kan de computer bewerkingen uitvoeren. Voorbeelden van computerprogramma's zijn: besturingssystemen, assemblers, compilers, interpreters, utilities, sorteerprogramma's, programma's voor gegevensbeheer, tekstverwerking, loonadministratie, voorraadbeheer en technische analyses.

Programmalus
Een aantal instructies die worden herhaald tot het gewenste resultaat is bereikt of tot de opgegeven eindstand van de tellervariabele is bereikt. Een lus verhindert dat u een instructie een groot aantal malen zou moeten gebruiken. Programmalussen behoren tot de meest gebruikte programmeertechnieken.

Prompt
Een of meer tekens op het scherm (bijvoorbeeld A>), die de gebruiker duidelijk maken dat het systeem wacht op ingetikte informatie. Een prompt wordt ook wel "systeemvraag" genoemd. U kunt ook zelf prompts opgeven, bijvoorbeeld in INPUT-opdrachten.

R

Radiaal
Eenheid voor het uitdrukken van de grootte van een hoek. Hoek aan het middelpunt van een cirkel waarvan de booglengte gelijk is aan de straal daarvan. Een rechte hoek (90°) is gelijk aan 1/2 pi radialen.

RAM
Acroniem van "Random Access Memory".

Randapparaat
Een extern apparaat bedoeld voor invoer/uitvoer of opslag van gegevens (bijvoorbeeld diskette-station, printer).

Random Access Memory (RAM)
Ook wel "werkgeheugen" genoemd. Dit is een geheugen voor snel opslaan en ophalen van gegevens. De gegevens in dit geheugen gaan verloren zodra de computer wordt uitgeschakeld.

Raster
Op een grafisch beeldscherm geeft een raster de horizontale of verticale afstand aan tussen twee adresseerbare punten op het scherm.

Read Only Memory (ROM)
Een geheugen met permanent opgeslagen gegevens of instructies. De computer kan de gegevens in dit geheugen lezen, maar kan er geen informatie in opslaan.

Real time
Wordt vrijwel nooit vertaald, maar soms aangeduid als "onvertraagd".
1. De tijd die nodig is voor het oplossen van een probleem of verwerken van instructies.

2. Het zo snel verwerken van instructies dat de resultaten daarvan als invoer kunnen dienen voor een volgende bewerking.

Reverse video
Gemarkeerde weergave d.m.v. het omwisselen van de kleuren voor voor- en achtergrond op het beeldscherm. Deze term wordt meestal gebruikt bij zwart/wit weergave.

ROM
Acroniem van "Read Only Memory".

RS-232
Een standaard communicatie-interface dat gebruikt wordt tussen een modem en werkstations of computers en dat voldoet aan EIA Standard RS-232.

S

Scheidingsteken
Een teken dat het begin of einde aanduidt van een gegevenseenheid, zoals een getal of tekenreeks, in het geheugen of in een bestand. Voor het onderling scheiden van deze eenheden worden tekens gebruikt als komma's, puntkomma's, punten en spaties.

Schrijfbeschermingsopening
Een inkeping in de hoes van de diskette. Als deze opening wordt afgedekt, kan geen informatie naar de diskette worden geschreven, maar kan de opgeslagen informatie wel worden gelezen.

Seriële uitvoer
Informatie versturen per bit. Dit werkt trager dan parallel I/O, waarbij 8 bits tegelijk worden verstuurd.

Single-density (Enkele dichtheid)
De standaard opnamedichtheid van een diskette. Op diskettes met single-density kunnen ongeveer 3400 bits (bpi) per inch worden opgeslagen.

Single-sided (Enkelzijdig)
Bij een single-sided diskette is slechts één zijde geschikt voor het opslaan van gegevens.

Software
Engelse term voor programmatuur. Instructies met behulp waarvan de computer bepaalde bewerkingen uitvoert.

Spoor
Een bepaald gebied op een geheugenmedium, zoals een diskette, schijf of cassette.

Stack
Ook wel "stapelgeheugen" genoemd. Opzet waarbij elk gedeelte van het geheugen kan dienen als "last-in", "first-out" stack voor het opslaan en oproepen van de inhoud van de accumulator, de flags (vlaggen) en data-registers. Veel eenheden bevatten een 16-bit stack pointer voor de adressering van deze externe stack. Een van de belangrijkste voordelen van de stack is dat meervoudige interrupts op eenvoudige wijze worden afgehandeld. Bij het optreden van een interrupt wordt de complete systeemstatus namelijk opgeslagen en weer hersteld na de interrupt. Een ander voordeel is dat een bijna onbeperkt aantal subroutines kunnen worden genest.

Synchroon
1. Bewerkingen waarvan de uitvoering wordt gestuurd door de pulsen van de interne klok in een computersysteem.

2. De klokpulsen volgen elkaar op met een steeds gelijkblijvend interval. Een synchrone bewerking kan oponthoud veroorzaken doordat wordt gewacht op een puls nadat de eerste bewerking al is afgesloten. De daaropvolgende bewerking begint echter pas na de eerstvolgende puls in plaats van onmiddellijk op de vorige aan te sluiten.

Syntaxis
Regels waaraan de structuur van een opdracht in een programmeertaal moet voldoen.

Standaard
De aanduiding voor een bewerking, commando of waarde die door het systeem automatisch wordt gekozen als de gebruiker zelf geen andere opgeeft.

T

Teken
Elk symbool dat de computer kan lezen, schrijven of opslaan zoals de letters van het alfabet, cijfers, tekens gebruikt voor interpunctie. Het woord "teken" is een synoniem voor "byte".

Toegang
Het zoeken, lezen of schrijven van gegevens op een geheugeneenheid.

Toepassingsprogramma (applicatieprogramma)
Een programma dat wordt gebruikt om met de computer een bepaalde taak te verrichten of een probleem op te lossen. Er wordt vaak onderscheid gemaakt tussen twee soorten toepassingen: "rekenprogramma's" en "programma's voor gegevensverwerking", waartoe toepassingen behoren als data-banken, tekstverwerking e.d.

U

Uitbreiden
Het uitbouwen van een systeem door opties, nieuwe software of randapparatuur toe te voegen.

Uitvoer
1. Het proces voor het uitvoeren van gegevens vanuit de computer (naar een randapparaat of diskette).
2. De gegevens die worden uitgevoerd.

Utility
Computerprogramma's met een bepaalde functie, die dienen als ondersteuning bij het gebruik van de computer. De functie FDISK is bijvoorbeeld bedoeld voor het opstellen van partities op de vaste schijf.

V

Variabele
Een grootheid die verschillende waarden kan aannemen, direct toegekend door de gebruiker of als resultaat van andere bewerkingen.

Vaste schijf
Wordt ook wel aangeduid met "hard disk" of "fixed disk". Een schijf die zich bevindt in een permanent gesloten behuizing, zodat de schijf is beschermd tegen invloeden van buitenaf. Deze schijf wordt gebruikt voor het opslaan van gegevens.

Veld
Een gebied dat in een record is gereserveerd voor een bepaalde categorie gegevens.

Verborgen bestanden
Bestanden die wel bestaan, maar niet in een directory zijn opgenomen.

Voeding
De schakeling die zorgt voor het transformeren van de netspanning naar de gelijkspanning voor de hardware van de computer.

Volume label
De naam die wordt toegekend aan de inhoud van een diskette of partitie op een vaste schijf.

Voorwaardelijke beslissing
Op grond van een vergelijking tussen een waarde of status en een vastgestelde voorwaarde het verloop van het programma sturen. Daarbij wordt vaak gebruik gemaakt van grenswaarden, (groter/kleiner dan).

W

Wissen
Het verwijderen van gegevens uit een bestand of van een opslagmedium.

Woord
Aantal bits die samen de grootste eenheid vormen die de computer in één keer kan verwerken. Een computer-woord is meestal gelijk aan een byte.

REGISTER

Andere BASICs 227
Apparaat niet beschikbaar 211, 213
ASCII-codes 216
Assembleertaal 219
Aanhalingstekens (") 99
ABS functie 48
Achtergrondkleur 62
Aftrekken 41
ASC functie 48
ASCII programmabestanden 121
Assembleerroutine 73
ATN functie 49
AUTO commando 49

Backspace 25
BACKSPACE toets 22
BASICs omzetten 227
BEEP opdracht 50
?Begin invoer opnieuw 99
Bestand bestaat reeds 212
Bestand niet gevonden 212
Bestand reeds open 108
Bestand reeds open 212
Bestandseinde gepasseerd 212
Bit complement 45
Bit-manipulaties 43
BLOAD opdracht 50
Booleaanse bewerkingen 43
Boomstructuur 87
BSAVE commando 51

CALL 220
CALL opdracht 52, 219
Cartesiaanse coördinaten 205
CD (Carrier Detect) 143
CDBL functie 54
CGA 148
CHAIN opdracht 55
CHDIR commando 56
Child-process 81, 182
CHR$ functie 56
CINT functie 57
CIRCLE opdracht 57
CLEAR commando 59
CLOSE opdracht 60
CLS opdracht 60
COLOR 157
COLOR opdracht 61
COM(n) opdracht 63
COM-detectie 127
Commando's 16
Commando-modus 12
COMMON opdracht 64
Communicatie 229
Communicatiebuffer vol 213
2's complement (binair) 96
Constante met dubbele precisie 37
Constante met enkele precisie 37
Constanten 36, 68
Control-codes 104
COS functie 65
CSNG functie 66
CSRLIN variabele 66
CTRL - 25
CTRL [..... 26
CTRL \ 25
CTRL] 25
CTRL-6 26
CTRL-ALT-DEL 98
CTRL-B 25
CTRL-BACKSPACE 26
CTRL-BREAK 25, 98
CTRL-C 25
CTRL-CURSOR-LINKS 25
CTRL-CURSOR-RECHTS 25
CTRL-E 26
CTRL-END 26
CTRL-ENTER 26
CTRL-F 25
CTRL-G 26
CTRL-H 25
CTRL-HOME 26
CTRL-I 27
CTRL-J 26
CTRL-K 26
CTRL-L 26
CTRL-M 26
CTRL-N 26
CTRL-NUMLOCK 27, 98
CTRL-PRTSC 27, 98
CTRL-R 26
CTRL-S 27
CTS (Clear To Send) 143
CURSOR LINKS 22, 25
CURSOR OMHOOG 22, 26
CURSOR OMLAAG 22, 25
CURSOR RECHTS 22, 25
CVI, CVS, CVD functies 67

DATA opdracht 67
Data-bestanden 29
DATE$ opdracht en variabele 69
Decimaal getal 36
DEF FN opdracht 69
DEF SEG 150
DEF SEG opdracht 72
DEF USR opdracht 73
DEFINT, DEFSNG, DEFDBL, DEFSTR opdrachten 71
DEL 26
Delen / 41
Delete (DEL) toets 22
DELETE commando 73
Deling door nul 210
Device Header "word attributes" 83
DIM opdracht 74
Directe opdracht gevonden 213
DRAW 153
DRAW opdracht 75
DSR (Data Set Ready) 143
Dubbele definitie 210
Dubbele precisie 71
Dubbele punt (:) 17
Dummy-argumenten 90

Edit 17
EDIT commando 78
Eenheid niet beschikbaar 143
EGA 147, 148
END 26
END opdracht 78
End-Of-File (EOF) 15
Enkele precisie 71
ENTER 26
ENVIRON opdracht 79
ENVIRON$ functie 80
EOF functie 82
ERASE opdracht 82
ERDEV, ERDEV$ variabelen 83
ERL 83, 131
ERR 83, 131
ERROR opdracht 83
ESC 26
EXP functie 85
EXTERR(n) functie 85
Eén-dimensionale matrix 39

False (onwaar) 43
FIELD opdracht 86
Field-lengte te groot 87, 212
File Control Block (FCB) 13
FILES commando 87
FIX functie 88
Floating Point Accumulator 224
FOR zonder NEXT 93, 130, 211
FOR...NEXT opdrachten 88
Foutmeldingen 209
FRE functie 90
Functie in uitgebreide BASIC 214
Functietoetsen 21, 27
Functionele operatoren 45

Geen DATA meer 95, 209
Geen geheugenruimte meer 210
Geen papier 83, 211
Geen RESUME 211
Geen ruimte meer voor karakterreeksen 59, 210
Geheel getal 36, 71
Geheugenadres 50
Geheugenmedium 51
Gereserveerde woorden 15
GET opdracht (Bestanden I/O) 91
GET opdracht (Grafisch) 91
Getal met decimale punt 36
GML-commando's 75
GOSUB...RETURN opdrachten 93
GOTO opdracht 95
Grafisch venster 61
Grafische bewegingen 75
Grafische coördinaat 75
Graphics Macro Language (GML) 75

Hernoemd naar andere schijf 214
HEX$ functie 95
Hexadecimaal getal 37
Hoge resolutie 58
HOME 26

IF opdracht 96
INKEY$ variabele 97
INP functie 98
INPUT opdracht 99
INPUT# opdracht 100
INPUT$ functie 101
INS 26
Insert (ins) toets 22
INSTR functie 102
INT functie 103
Integer-delen 42
Interne fout 212
Invoerbuffer 115
Invoer/Uitvoer fout 212
I/O Communicatie 229
IOCTL opdracht 104
IOCTL$ functie 104

Kader 62
Karakterreeks te complex 210
Karakterreeks te lang 210
Kan programma niet vervolgen 210
KEY 127
KEY opdracht 105
KEY(n) opdracht 107
KILL commando 107

LEFT$ functie 108
Legato 152
LEN functie 108
LET opdracht 109
Line clipping 110
LINE INPUT opdracht 112
LINE INPUT# opdracht 113

LINE opdracht 110
Line-styling 110
LIST commando 113
Listing 114
LLIST commando 114
LOAD commando 115
LOC functie 115
LOCATE opdracht 116
LOCK opdrachten 117
LOF functie 118
LOG functie 119
Logische operatoren 40, 43
Loop 88, 202
LPOS functie 119
LPRINT en LPRINT USING opdrachten 120
LSET en RSET opdrachten 120
Lusbewerking 89

Machinetaal 219
Machtsverheffen 41
Matrix 39
Matrix-dimensie 39
Matrix-variabelen 64
Meerdimensionale matrices 39
Meldingen 209
MERGE commando 121
MID$ functie 121
MID$ opdracht 122
MKDIR commando 123
MKI$, MKS$, MKD$ functies 123
Modulo 42

NAME commando 124
Negatief maken - 41
Nesten 89
Netwerken 117
Neutrale tekens 87
NEW commando 124
NEXT zonder FOR 89, 132, 209
Niet afdrukbare fout 211, 212, 213
Nul-string 80
Numerieke argumenten 77
Numerieke constanten 36
Numerieke functies 16
Numerieke toetsen 25
Numerieke waarden 67, 101

OCT$ functie 125
Octaaf 153
Octaal getal 37
ON ...(n) opdrachten 125
ON COM opdracht 129
ON COM(n) opdracht 129
ON ERROR GOTO opdracht 130
ON KEY(n) opdracht 132
ON PEN opdracht 134
ON PLAY opdracht 135
ON STRIG(n) opdracht 136
ON TIMER(n) opdracht 137
ON...GOTO en ON...GOSUB opdrachten 131
Onbepaald benaderbare bestanden ... 31, 86
Onderbroken in regel xxxx 78
Ongedefinieerd regelnummer ... 78, 96, 210
Ongedefinieerde gebruikersfunctie 70, 211
Ongeldig in directe modus 210
Ongeldige bestandsnaam 213
Onjuist apparaat 211
Onjuist bestandsnummer 51, 212
Onjuist bloknummer 213
Onjuist type 68, 210
Onjuist bestandsnummer 140, 212
Onjuiste functieaanroep 48, 103, 209
Ontbrekende operand 211
Onwaar (false) 43
Opdrachten 16
OPEN COM(n) opdracht 142
OPEN opdracht 138
Opslaan van een bestand 23
Opslagruimte 67
Optellen 41
OPTION BASE opdracht 144
OUT opdracht 144
Overlay 55

Pad niet gevonden 214
Pad/bestand niet toegankelijk 142, 214
Paint "tiling" 145
PAINT opdracht 145
Palet 62
PALETTE 157
PALETTE opdracht 147
PALETTE USING opdracht 147
Pariteitscontrole 143
PCOPY commando 149
PEEK 156
PEEK functie 150
PEN functie 150
PEN OFF 134, 150
PEN ON 134, 150
PEN opdracht 150
PEN STOP 134, 150
Pieptoon 50
Pijlsymbool <- 106
Pixels 92
PLAY OFF 135
PLAY ON 135
PLAY opdracht 152
PLAY STOP 135
PLAY(n) 128
PLAY(n) functie 153
Plusteken (+) 46
PMAP functie (Grafisch) 154
POINT functie 154
POKE 156
POKE opdracht 155
POS functie 156
PRESET opdracht 156
PRINT opdracht 158
PRINT USING opdracht 159
PRINT# opdracht 162
PRINT# USING opdracht 162
Programma-modus 12

Programmabestanden 28
PRTSC 98
PSET opdracht 156
PUT opdracht (Bestanden) 165
PUT opdracht (Grafisch) 164

Random access files 86
RANDOMIZE opdracht 166
READ opdracht 168
Records 31
Regelbuffer vol 211
Regels wijzigen 22
Relationele operatoren 43
REM opdracht 169
RENUM commando 170
Reset 133
RESET commando 171
RESTORE commando 171
RESUME opdracht 172
RESUME zonder fout 211
RETURN opdracht 172
RETURN zonder GOSUB 209
RIGHT$ functie 173
RMDIR functie 173
RND functie 174
RTS (Request To Send) 143
RUN commando 174

SAVE commando 175
Scherm-modi 61
Schermbuffer 51
Schijf heeft schrijfbeveiliging 214
Schijf is vol 212
Schijf niet gereed 214
Schijffout 214
Schuine streep (\) 42
SCREEN functie 176
SCREEN opdracht 177
Seed 166, 174
Segmentadres 72
Sequentiële bestanden 29
SGN functie 181
SHELL opdracht 182
Shift-bewerking 54
SHIFT-PRTSC 27
SHIFT-toetsen 105
SIN functie 183
SOUND opdracht 183
SPACE$ functie 185
SPC functie 185
SQR functie 186
Staccato 153
Stack 145
Stack-ruimte 59
Sterretje (*) 49, 87
STICK functie 186
STOP opdracht 187
STR$ functie 188
STRIG opdracht en functie 188
STRIG(n) 128
STRIG(n) opdracht 189
String descriptor 53
STRING$ 158
STRING$ functie 190
Subdirectories 87
Subroutine 52, 219
SWAP opdracht 190
Syntaxisfout 130, 209
SYSTEM commando 191

Tab 27
TAB functie 191
TAN functie 192
Te groot getal 43, 57, 209
Te hoog indexgetal 74, 210
Te veel bestanden 213
Tekenreeks 71
Tekenreeks te lang 84
Tekenreeks-constante 36
Tekenreeks-expressie 51
Tekenreeks-functies 16
Tekenreeks-operatoren 45
Tekenreeks-ruimte 53
Tekenreeks-vergelijkingen 46
Tekenreeksen 67, 101
Tekens voor type-aanduidingen 38
Tekst-tekenreeksen testen op gelijkheid 53
Tempo 153
Tiles 145
TIME$ opdracht en variabele 192
TIMER functie 193
TIMER(n) 128
Toetscodes 98
Toetscombinaties 25
Toetsen 25
Tokens 15
Trace off 194
Trace on 194
TRON en TROFF commando's 194
True (waar) 43
TTY-voorbeeldprogramma 231
Twee-dimensionale matrix 39
Two's complement 45
Type conversie 40
Type-aanduidingen 38

UNLOCK opdracht 194
USR functie 196, 219, 223

VAL functie 196
Variabelen 16, 37
VARPTR 219
VARPTR functie 197
VARPTR$ functie 199
Vermenigvuldigen 41
VIEW 154
VIEW opdracht 200
VIEW PRINT opdracht 201
Voorbeelden assembler 225
Voorgrondkleur 62
Vraagtekens (?) 87

Waar (true) 43

WAIT opdracht 201
WEND zonder WHILE 211
Wereldcoördinaten 205
Wetenschappelijke notatie 36
WHILE zonder WEND 211
WHILE...WEND opdracht 202
WIDTH opdracht 203
Wijzigen 24
WINDOW 154
WINDOW opdracht 205
Wiskundige functies 215
Wiskundige operatoren 41
WRITE opdracht 206
WRITE# opdracht 207

Zelf gedefinieerde fouten 84
Zelf gedefinieerde functies 16

GEBRUIKERSCOMMENTAAR

72-10251-01/331

Door het invullen van dit formulier en het opsturen ervan naar Tulip Computers helpt u ons met het verbeteren van onze produktdocumentatie.

Datum: _____

Welke gedeeltes van deze handleiding waren het best bruikbaar?

Hebt u suggesties voor verbetering van deze documentatie? Ontbreekt er belangrijke informatie? Zijn er meer voorbeelden nodig? Zo ja, waar?

Trof u fouten in deze handleiding aan? Op welke pagina('s)?

Stuur dit formulier naar:
Tulip Computers
Afdeling documentatie
Postbus 3337
5203 DH 's-Hertogenbosch
Nederland

Uw naam:

Firma:

Adres:

Gebruikerscommentaar

LICENSE AGREEMENT

BY OPENING THIS PACKAGE YOU ARE AGREEING TO BECOME BOUND BY THE TERMS OF THIS AGREEMENT, WHICH INCLUDES THE SOFTWARE LICENSE, SOFTWARE DISCLAIMER OF WARRANTY, AND HARDWARE LIMITED WARRANTY (collectively the "Agreement"). THIS AGREEMENT CONSTITUTES THE COMPLETE AGREEMENT BETWEEN YOU AND MICROSOFT CORPORATION.

IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, RETURN THE UNOPENED DISK PACKAGE AND THE OTHER ITEMS (INCLUDING WRITTEN MATERIALS, BINDERS OR OTHER CONTAINERS, AND HARDWARE, IF ANY) THAT ARE PART OF THIS PRODUCT TO THE PLACE WHERE YOU OBTAINED THEM FOR A FULL REFUND.

MICROSOFT SOFTWARE LICENSE

1. GRANT OF LICENSE. In consideration of payment of the LICENSE fee, which is a part of the price you paid for this product, Microsoft, as Licensor, grants to you, the LICENSEE, a nonexclusive right to use and display this copy of a Microsoft software program (hereinafter the "SOFTWARE") on a single COMPUTER (i.e. with a single CPU) at a single location. If the single computer on which you use the SOFTWARE is a multiuser system, the License covers all users on that single system. Microsoft reserves all rights not expressly granted to LICENSEE.

2. OWNERSHIP OF SOFTWARE. As the LICENSEE, you own the magnetic or other physical media on which the SOFTWARE is originally or subsequently recorded or fixed, but Microsoft retains title and ownership of the SOFTWARE recorded on the original disk copy(ies) and all subsequent copies of the SOFTWARE, regardless of the form or media in or on which the original and other copies may exist. This License is not a sale of the original SOFTWARE or any copy.

3. COPY RESTRICTIONS. This SOFTWARE and the accompanying written materials are copyrighted. Unauthorized copying of the SOFTWARE, including SOFTWARE that has been modified, merged, or included with other software, or of the written materials is expressly forbidden. You may be held legally responsible for any copyright infringement that is caused or encouraged by your failure to abide by the terms of this License. Subject to these restrictions and if the SOFTWARE is not copy-protected, you may MAKE 1 (ONE) COPY OF THE SOFTWARE SOLELY FOR BACKUP PURPOSES. You must reproduce and include the copyright notice on the backup copy.

4. USE RESTRICTIONS. As the LICENSEE, you may phisically transfer the SOFTWARE from one computer to another provided that the SOFTWARE is used on only one computer at a time. You may not electronically transfer the SOFTWARE from one computer to another over a network. You may not distribute copies of the SOFTWARE or accompanying written materials to others. You may not modify, adapt, translate, reverse engineer, decompile, disassemble, or create derivative works based on the SOFTWARE. You may not modify, adapt, translate, or create derivative works based on the written materials without the prior written consent of Microsoft.

5. TRANSFER RESTRICTIONS. This SOFTWARE is licensed only to you, the LICENSEE, and may not be transferred to anyone without the prior written consent of Microsoft. Any authorized tranferee of the SOFTWARE shall be bound by the terms and conditions of this Agreement. In no event may you transfer, assign, rent, lease, sell, or otherwise dispose of the SOFTWARE on a temporary or permanent basis except as expressly provided herein.

6. TERMINATION. This License is effective until terminated. This License will terminate automatically without notice from Microsoft if you fail to comply with any provision of this License. Upon termination you shall destroy the written materials and all copies of the SOFTWARE, including modified copies, if any.

7. UPDATE POLICY. Microsoft may create, from time to time, updated versions of this SOFTWARE. At its option, Microsoft will make such updates available to the LICENSEE and transferees who have paid the update fee and returned the Registration Card to Microsoft.

8. MISCELLANEOUS. This Agreement is governed by the laws of the State of Washington.

DISCLAIMER OF WARRANTY AND LIMITED WARRANTY

THE SOFTWARE AND ACCOMPANYING WRITTEN MATERIALS (INCLUDING INSTRUCTIONS FOR USE) ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. FURTHER, MICROSOFT DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REAGARDING THE USE, OR THE RESULTS OF THE USE OF THE SOFTWARE OR THE WRITTEN MATERIALS IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. IF THE SOFTWARE OR WRITTEN MATERIALS ARE DEFECTIVE YOU, AND NOT MICROSOFT OR ITS DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES, ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR, OR CORRECTION.

Microsoft warrants to the original LICENSEE that (a) the disk(s) on which the SOFTWARE is recorded is free from defects in materials and workmanship under normal use and service for a period of 90 (ninety) days from the date of delivery as evidenced by a copy of the receipt and (b) the hardware accompanying the SOFTWARE is free from defects in materials and workmanship under normal use and service for a period of 1 (one) year from the date of delivery as evidenced by a copy of the receipt.

Microsoft's entire liability and your exclusive remedy as to the disk(s) or hardware shall be, at Microsoft's option, either (a) return of the purchase price or (b) replacement of the disk or hardware that does not meet Microsoft's Limited Warranty and which is returned to Microsoft with a copy of the receipt. If failure of the disk or hardware has resulted from accident, abuse, or misapplication, Microsoft shall have no responsibility to replace the disk or hardware or refund the purchase price. Any replacement disk or hardware will be warranted for the remainder of the original warranty period or 30 (thirty) days, whichever is longer.

THE ABOVE ARE THE ONLY WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, THAT ARE MADE BY MICROSOFT ON THIS MICROSOFT PRODUCT. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY MICROSOFT, IT'S DEALERS, DISTRIBUTORS, AGENTS, OR EMPLOYEES SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY, AND YOU MAY NOT RELY ON ANY SUCH INFORMATION OR ADVICE.

NEITHER MICROSOFT NOR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THIS PRODUCT SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, OR INCIDENTAL DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OF OR INABILITY TO USE SUCH PRODUCT EVEN IF MICROSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This disclaimer of warranty is governed by the laws of the State of Washington.

ACKNOWLEDGEMENT

YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LICENSE AND LIMITED WARRANTY, UNDERSTAND THEM, AND AGREE TO BE BOUND BY THEIR TERMS AND CONDITIONS. YOU ALSO AGREE THAT THE LICENSE AND LIMITED WARRANTY ARE THE COMPLETE AND EXCLUSIVE STATEMENT OF AGREEMENT BETWEEN THE PARTIES AND SUPERCEDE ALL PROPOSALS OR PRIOR AGREEMENTS ORAL OR WRITTEN, AND ANY OTHER COMMUNICATIONS BETWEEN THE PARTIES RELATING TO THE SUBJECT MATTER OF THE LICENSE OR THE LIMITED WARRANTY.

Should you have any questions concerning this Agreement, or if you desire to contact Microsoft for any reason, please contact in writing: Microsoft Customer Sales and Service / 10700 Northup Way / Box 97200 / Bellevue, WA 98009 / U.S.A.